Freie Universität Bozen
Libera Università di Bolzano
Free University of Bozen · Bolzano

Fakultät für Informatik | Facoltà di Scienze e Tecnologie informatiche | Faculty of Computer Science

Hierarchical Summarization of Multidimensional Data

Tesi per il Dottorato di Ricerca in Informatica

Doktorarbeit in Informatik

PhD Thesis in Computer Science

Andrej Taliun

Relatore - *Doktorvater* - Faculty Advisor: Prof. Michael H. Böhlen
Correlatore - *Zweitbetreuer* - Faculty Co-Advisor: Dr. Arturas Mazeika

ACM categories: Data Structures (E.1), Information Search and Retrieval (H.3.3), Clustering (I.5.3)

For My Family

Contents

List of Figures

List of Tables

Acknowledgments

In the first place, I express my gratefulness to supervisors Michael H. Böhlen and Arturas Mazeika. During my Ph.D. They in all ways supported my ideas, pushed me towards strong results and always gave very valuable and helpful feedback. Michael H. Böhlen and Arturas Mazeika stayed always close to my work and in all directions helped to progress it. The regular meetings and discussions kept gradually improving my research abilities and communication skills. Michael H. Böhlen and Arturas Mazeika taught me how to write research papers, develop the ideas and that it takes to achieve good research. I want especially thank my supervisors for many iterations that they made to improve my papers.

From the whole of my heart, I thank my lovely wife Diana for the unlimited support in my work. She took all my responsibilities and duties and always encouraged me with her honesty, kindness and infinite love. Without that, it would be not possible for me to fully devote myself on finishing the Thesis.

I also thank my Parents for their love, encouragement and support throughout the all the years.

I acknowledge my gratitude to my colleagues. Especially, I thank Romans Kasperovics for patiently listening to my ideas and valuable criticism.

Abstract

Data summarization and clustering are key techniques to query and analyze large amounts of multidimensional data. However, the effectiveness of existing methods is limited by high intermediate memory cost and difficult to choose input parameters. This Ph.D. thesis develops a novel approach to hierarchical data summarization and clustering that overcomes these limitations.

We propose the AD-Tree, an innovative data summary structure that summarizes multidimensional data in terms of its density on a hierarchy of grids. The computation of the AD-Tree is an iterative process that requires a minimal amount of intermediate memory. We start computing the AD-Tree from a sparse initial grid, which gives a rough estimate of the density function of the data. We iteratively increase the estimation quality by splitting cells along dimensions where the density function is non-linear. This ensures a minimal consumption of intermediate memory: instead of overestimating the density on a fine grid and afterwards removing unnecessary grid points, we put new grid points only in places where it increases the precision of the estimation. The key challenges of our approach are the identification of areas and dimensions where the density function exhibits a non-linear behavior and a fast organization of new grid points into a hierarchy of grids that ensures an optimal memory utilization. We introduce *shape error*, *dimensional split*, *grouping* and *compact representation* of multidimensional grids to successfully solve these problems: the shape error measures the deviation of the density function from being linear on a grid, the dimensional split implements the splitting of cells in selected dimensions, the grouping organizes new grid points into large grids, and the compact representation of multidimensional grids reduces their storage costs by a factor of the dimensionality. We develop an efficient solution to approximately answer aggregate range queries from the AD-Tree.

We develop CORE, a novel clustering technique that clusters multidimensional data without any input parameters. The salient property of CORE is the explicit computation and representation of local density maxima, which permits

a high-quality nonparametric clustering. CORE uses the local density maxima to determine cores of clusters. The AD-Tree, *rectangular neighborhoods* and *gradients* enable the efficient and robust computation of cores: the AD-Tree provides a uniform and compact estimation of the density of the data, the rectangular neighborhood localizes stationary points in the AD-Tree, and gradients distinguish local maxima from other types of stationary points and connect maximal cores. CORE is the first clustering technique that bases the clustering on a semantically rich data summary structure.

We investigate overlapping clusters and develop an efficient solution to separate them. The separation of overlapping clusters makes it necessary to cluster the data at all levels of the density and to consider the orientation of clusters. We use the AD-Tree, which allows CORE to find fragments and overlapping cores at all levels of the density. We restore complete cores from their fragments with the help of *gradient paths*. Gradient paths connect fragments through overlaps and quantify the orientation of fragments.

We analytically investigate our techniques and confirm the results with extensive experimental evaluations on synthetic and real world datasets. The results show the advantage of our techniques compared to existing methods.

Chapter 1

Introduction

The last two decades witnessed a tremendous increase of the amounts and especially the dimensionality of the data. Data summarization, approximate query answering and clustering are the standard techniques to handle this data. The limitations of current solutions are a high intermediate memory complexity to build and/or query the summary (although the size of the summary is small), many parameters that allow to tune the methods to achieve specific goods but are rather unintuitive and hard to adjust even for experts. This thesis develops a hierarchical summarization of multidimensional data and a set of associated techniques to query and mine multidimensional data with the help of this summary structure. Essential principals for a summary of multidimensional data are a minimal intermediate memory consumption, a good organization of multidimensional data due to limit the growth as the dimensionality increases, a low number and intuitive input parameters, and techniques to effectively use the summary structure to query and mine the data.

This thesis proposes the AD-Tree data summary for multidimensional data. The AD-Tree uses density information to summarize multidimensional data. Construction of the AD-Tree structure minimizes the required intermediate memory. The summarization of the density at grid points and the organization of grid points into a hierarchy of grids enables an effective storage of the summary. This is in strong contrast to other multidimensional data summarization techniques that first compute the density on a very fine grid and then compress the grid or remove the unneeded grid points. Such approaches become prohibitively expensive already for multidimensional data with low dimensionality ($d = 3$–4).

We show how to precisely and efficiently estimate answers of aggregate range queries from the AD-Tree. Aggregate range queries can be answered by computing multiple integrals of the density function estimated on the hierarchy of grids of the AD-Tree. The exact computation of multiple integrals of an arbitrary function is difficult and expensive. In our work we reduce the computation of multiple integrals to cheap linear interpolations with guarantees for precision of

the answer. A high precision of the estimated answers for aggregate queries is ensured by filtering out the overlaps between grids within a query range.

This thesis is the first work that successfully uses a high dimensional data summary structure for clustering. For a given precision ε the AD-Tree summarizes the density of the data and ensures that the quality of the estimation of the density is uniform within the whole domain of the database. The precision parameter ε exhibits a strictly monotonic behavior: a decrease of the precision monotonically increases the uniform estimation error of the density. Based on the AD-Tree we design and develop CORE, a non-parametric clustering technique. CORE models clusters with connected sets of points where the density is locally maximal(cores) and uses gradients of the density to assign dataset points to cores (labeling of the data). This makes the clustering invariant to noise and robust to local fluctuations of the density of the data.

The thesis investigates the separation of overlapping clusters, which is a novel direction towards improving the quality of clustering methods. Overlapping clusters occur in many real world datasets and their separation is difficult. In order to separate overlapping clusters a successful clustering technique must find local maxima at all density levels and distinguish between the local maxima of overlaps and fragments of overlapping clusters. The use of the AD-Tree data summary is the key for the successful separation of overlapping clusters by CORE. CORE finds local maxima at all levels of the density and uses gradients to represent orientation of clusters. This makes CORE a leading clustering technique, which can be used to separate overlapping clusters of any dimensionality.

In the rest of the introduction we list the specific contributions of the chapters of the thesis. The thesis is organized as a collection of self-contained chapters that can be read in isolation. They have been slightly modified to reduce overlap, synchronize notation and terminology, and ensure that the whole thesis can be read sequentially.

Chapter 2 defines the AD-Tree data summary structure. The salient feature of the AD-Tree is the iterative construction which requires no extra intermediate memory. The construction of the AD-Tree starts with a sparse initial grid and each subsequent iteration only adds new grid points if this increases the precision of the estimation. That is in contrast to other data summary structures which, first, overestimate the density on a fine partition and, later, remove unnecessary partition units. We quantify the estimation quality with the help of the *shape error* and *dimensional split*. The shape error measures the deviation of the density function from being linear on a one-dimensional grid. The dimensional split extends the shape error to multidimensional grids: it splits a d-dimensional cell only along the dimensions of nonlinearity of the density function and leaves other dimensions intact. The iteration finishes with the *grouping* step, which groups all newly introduced individual grid points into regular grids. The grouping step substantially speeds up the construction of the AD-Tree and optimizes the data structure for querying. Optimal grouping is hard and, hence, we de-

velop an efficient solution that runs in linear time and maximizes the size of the grids. We prove the effectiveness of the AD-Tree with the extensive analytical and experimental evaluations.

Since the AD-Tree is a continuous estimate of the density function it can be used to approximately answer aggregate range queries. Aggregate queries are expressed by multiple integrals over the cells in the query range. First, we show how to reduce the computation of multiple integrals to linear interpolations of the density at a single point of a cell. Next, we offer two algorithms to filter out unsplit cells in the query range. The algorithms have different best and worst case scenarios depending on the number of cells in the query range and number of child nodes.

CORE offers nonparametric model of clusters and overcomes other clustering methods in quality of computed clusters. Chapter 3 develops CORE, an efficient and non-parametric clustering technique. CORE explicitly computes the local maxima of the density and represent them with cores of clusters. The core of a cluster is a connected sets of local maxima points. The nonparametric computation of cores is hard due to fluctuations in the density. CORE uses *rectangular neighborhood* and *gradients* to robustly and nonparametrically compute the cores. For a given AD-Tree, the rectangular neighborhood localizes stationary points of the density. The directions of gradients are used to distinguish local maxima points from other types of stationary points and to combine individual local maxima points into connected sets. Algorithmically, CORE clusters the data in three steps. First, CORE computes the AD-Tree from a representative sample of the dataset. The AD-Tree provides an accurate estimation of the density of the data. Second, CORE computes cores of clusters with the help of the rectangular neighborhood and gradients. Third, CORE assigns data points to the cores of clusters. Each data point is assigned to a core based on the paths of gradients. A cluster consists of all data points assigned to the corresponding core. We give a thorough analytical and experimental evaluation of CORE and show that CORE accurately detects arbitrarily dense and shaped clusters in large datasets without requiring input parameters.

Chapter 4 extends CORE to the separation of overlapping clusters. Overlapping clusters are often present in time-varying data and their separation is difficult for the following reasons. First, overlaps accumulate the density of overlapping clusters and, thus, produce new density peaks. Second, overlaps divide clusters into fragments. There are two essential steps to successfully separate overlapping clusters. First, in order to separate cluster from the overlaps, we need to cluster the data at all level of the density and, second, for an accurate reconstruction of clusters from their fragments we need to consider their orientation.

The papers included in the thesis are listed below. Chapter 2 is based on Paper 1 and Paper 3. Chapter 3 is based on Paper 2. Chapter 4 is based on Paper 4.

1. Mazeika, A., Böhlen, M. H., and Taliun, A. *Adaptive density estimation.* In Proceedings of the *32nd International Conference on Very Large Data Bases*, 4 pages, Seoul, Korea, 2006.

2. Taliun, A., Böhlen, M. H., and Mazeika, A. *CORE: Nonparametric Clustering of Large Numeric Databases.* Proceedings of the *SIAM International Conference on Data Mining*, 12 pages, Sparks, USA, 2009.

3. Taliun, A., Böhlen, M. H., and Mazeika, A. *Hierarchical Approach for Fast Approximate Answering of Aggregate Range Queries*, to be submitted.

4. Taliun, A., Böhlen, M. H., and Mazeika, A. *Separation of Overlapping Clusters*, to be submitted.

Chapter 2

The AD-Tree

2.1 Introduction

The construction of summary structures for multidimensional data is a challenging task. For the most part, the database community focused on improving the computation speed, compactness, query time and approximation quality of summary structures. Much less efforts were made to minimize the use of intermediate memory needed to construct the summary structure. The common way to construct summary structures is, first, to overestimate density and, then, iteratively remove unnecessary partition units. For example, wavelet based techniques compute accumulated density on a large multidimensional grid and then compress it using discrete wavelet transform. Such an approach suffers from high consumptions of intermediate memory if the data has more than 5 or 6 dimensions (the memory consumption required to store a 5-dimensional regular grid of granularity 50 in each dimension is more than 1Gb).

In our work we develop the adaptive density tree (AD-Tree), a multidimensional data summary structure whose computation consumes no intermediate memory. Technically, the AD-Tree estimates the density function of the data and is based on the Kernel method. The computation of the AD-Tree starts from a sparse regular grid, which gives a rough estimation of the density function. Next, we iteratively add new partition points in areas of the grid and along dimensions where the precision of the density estimation increases. This reduces the memory complexity substantially: in contrast to a fine regular grid where all dimensions are equally partitioned, we put partition points only in sub-dimensions where the density of the data changes.

There are two key challenges in the AD-Tree method. The first challenge is to effectively and efficiently identify position and number of new partition points that will yield the highest increase of the estimation precision of the density function. This is a fundamental problem: in order to check whether new partition points increase the precision, we need a precise estimation of the density function.

The second challenge is to organize partition points in a data structure that yields a minimal memory utilization and supports fast querying. Our solution for that are *the shape error*, *dimensional split* and *bifurcation grid*. The shape error measures the deviation of the estimated density function from being linear on a one dimensional grid segment. We put new partition points only in areas of the grid where the shape error is high, i.e., where the density is non-linear. The dimensional split extends shape error to multiple dimensions: for each cell of a d-dimensional grid it identifies dimensions in which the shape error is high and places new partition points at intersections of splitting planes with faces of the cell. All new partition points are organized into a hierarchy of compact regular grids, which reduces the size of grid frames by a factor of d.

Summarizing, in our work we make the following contributions:

- We define the AD-Tree, a multidimensional data summary structure whose construction consumes no intermediate memory. The AD-Tree uses shape error and dimensional split to efficiently allocate new partition points within areas and along dimensions of nonlinearity of the density function of the data.

- We optimize the data structure of the AD-Tree for multidimensional data. The AD-Tree uses compact representation of grids for memory efficient storage of frames and granularity, and has only of one pointer per node that references all child.

- We investigate the AD-Tree analytically. We prove that the AD-Tree adjusts to the local dimensionality of the data and provides an optimal partitioning. In contrast to other space partitioning methods the AD-Tree organizes density information rather than the data, and the partitioning does not depend on the number of observations.

- We provide an in-depth empirical evaluation of our density estimator. The numerical evaluation details the time and space requirements for various precisions. We show that the AD-Tree adjusts to the dimensionality of the structures in the dataset. The method effectively reduces the memory complexity for databases containing structures of lower dimensionality than the dimensionality of the dataset.

The chapter is organized as follows. We give preliminaries on the kernel method and used notation in Section 2.3. We discuss related work in Section 2.2. We give the detail of the construction of the AD-Tree in Section 2.4. In Section 2.5 we define the data structure of the AD-Tree. Section 2.7 shows how the AD-Tree can be used for efficient computation of aggregate range queries. Analytical properties of the AD-Tree method are described in Section 2.6. Section 2.8 presents

algorithms for the construction of the AD-Tree and analyzes their complexity. Experimental evaluation is given in Section 2.9. Section 2.10 concludes the chapter and offers future work.

2.2 Related Work

In this section we discuss related work on data summarization for application of selectivity estimation and approximate query answering. In particular, we compare the AD-Tree with histograms and wavelets. Histograms and wavelets provide accurate data compression, however, suffer from high intermediate memory complexity.

[23] and [25] improve intermediate memory complexity for constructing optimal histograms and wavelets. Optimal histograms [30] [31] and wavelets [19] [15] aim to find the best size and position for given number of buckets/coefficients which give the smallest maximum error. Intermediate memory complexity of both optimal histograms and wavelets is quadratic wrt to n, where n is either the size of the dataset or the size of the temporal multidimensional array. Time complexity to compute optimal histograms and wavelets is linear and quadratic wrt n correspondingly. [23] offers a solution which computes optimal histograms and wavelets in linear intermediate memory and quadratic time wrt to n. Computation of the AD-Tree consumes no intermediate memory and is linear wrt to the size of the dataset. The AD-Tree minimizes *the shape error* and ensures uniform precision of the estimation.

2.2.1 Histograms

[62] proposes a *hierarchical model fitting histograms* which generalize other multidimensional histogram techniques. The key idea of [62] is to allow each bucket contain different type of summary information that is in contrast to previous solutions where all buckets contain the same type of summary information. Similarly to the AD-Tree, construction of the HFM is iterative: it starts with a single bucket which is partitioned into smaller buckets until memory constraints are reached. HFM choices partition dimension and the type of the summary information based on the minimal description length principle. Differently from the AD-Tree, computation of HFM requires additional space to store tuples of the dataset and is quadratic to the number of tuples in the dataset. The AD-Tree is computed in a linear time and does not require any additional memory.

In the PHASED histogram [46], a dimension is first chosen and the multidimensional array is split along that dimension into several buckets in which the sums of measure values are nearly equal. Each bucket is then treated as a (d-1)-dimensional array and is split recursively for the remaining dimensions. Typically, the order in which the dimensions are to be split is decided arbitrary

and the number of splits along each dimension are the same. Although [46] is constructed with no use of intermediate memory, it was shown that the quality and query speed of PHASED histogram is much worst than other multidimensional histograms [26], [6] and [62]. Construction of PHASED histogram is based on a simple heuristic which does not consider local dimensionality of structures in the dataset and does not guarantee a uniform estimation error. The AD-Tree adopts to local dimensionality of structures, ensures uniform estimation error and organizes grid points into a hierarchy of grids for fast querying.

[61] improves PHASED histogram in terms of quality and computation time with an additional scanning step. During that step a temporal multidimensional equi-width histogram is computed and, later on, PHASED histogram is build from a temporal histogram rather than from the initial relation. [61] has high cost of intermediate memory since, multi-dimensional equi-width histogram of high precision does not fit in the main memory already for multidimensional data of low dimensionality.

GenHist [26] permits overlapping buckets in areas where the data distribution is nonuniform, and approximates the density of an area with overlapping buckets as the sum of the densities of the overlapping buckets. Overlapping buckets allow GenHist to achieve a compact representation of the data distribution [1, 50, 60]. However, overlapping buckets slow down the query answering because all buckets that overlap the query range must be considered, even if their contribution to the overlap is small. Similarly to [61], GenHist has high intermediate memory complexity since it uses temporal multidimensional equi-width histogram for computation of overlapping buckets. In contrast to the AD-Tree and optimal histograms, GenHist does not minimize any error metric and does not guarantee uniform precision of the estimation.

STHoles [6] is a multidimensional histogram that allocates buckets based on query results. STHoles achieves a good precision and is robust to changes in the workload. On average, STHole histograms are as precise as GenHist histograms for the same memory. Similar to GenHist, STHoles allows overlapping buckets by partitioning a large (parent) bucket into a number of smaller (child) buckets, and stores buckets in a tree like structure. This yields a faster response time than scanning all buckets. Differently from the AD-Tree, STHoles stores buckets individually and allocates $3d$ digits per bucket which describe its position and size. Thus, STHoles is less memory efficient than the AD-Tree. The AD-Tree reduces representation of grids by a factor of d. [42] improves [6] by using discrete approximation of position and size of the buckets, however, in cost of compression quality.

[16] proposes a solution which uses one-dimensional histograms to summarize multidimensional data. The key idea of [16] includes two steps: i) compute statistical interaction model that identifies (to some given error) correlated low-dimensional subsets of data tuple and ii) each low-dimensional subset approximate with one-dimensional histogram. [16] has worst quality and higher

computation complexity compared to multidimensional histograms.

[52] designs histograms for efficient answering of aggregate group by queries on streams of unique identifiers (such as IP addresses). These streams involves partitioning into groups using large lookup tables. The authors discuss the following scenario: the control center computes a partitioning function (i.e. initial histogram) using a lookup table and send its copy to each monitor in the network; each monitor refines a copy of the histogram and send it back the center; later all copies are merged into one histogram and aggregate queries are approximated. [52] proposes three algorithms for computation of histograms from lookup tables which produce more accurate results that traditional histograms for streams of unique identifiers.

Computations of multidimensional histograms on data streams so that at any time histograms reflects current data are studied in [58]. As a solution the authors of [58] propose to maintain *s*ketches, a *d*-dimensional vectors which a are defined as multiplication of last data records in the stream with random linear mapping. The authors provide several algorithms to compute multidimensional histogram from the sketch. Update operation on sketches are constant in time that make them practical and desirable in data stream applications. The shortcoming of the approach is exponential time for computing histograms from sketches.

2.2.2 Waveletes

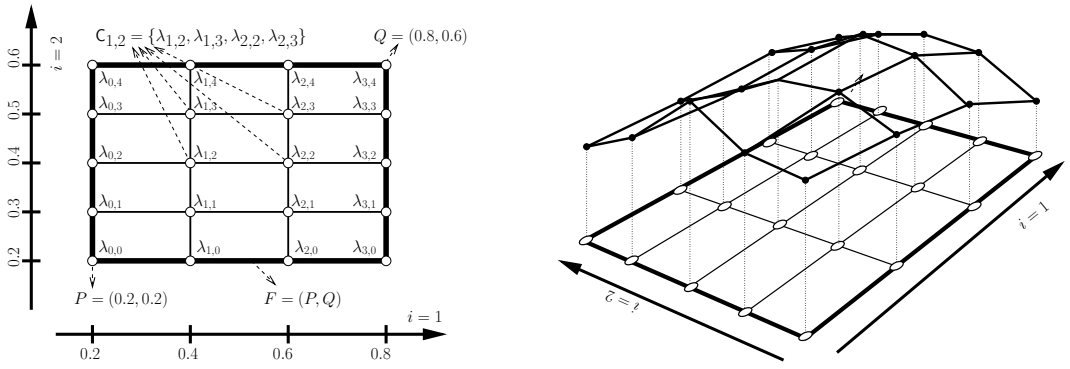
Wavelets [43, 60, 27] compress density information to get small summaries. First, wavelet techniques use standard histogram techniques to compute the extended cumulative data distribution. Second, the histograms are compressed using discrete wavelet transform. The density value at a given point is approximated by decompressing $2d$ buckets. The decompression step is expensive since all wavelets coefficients are involved in the computation. Moreover, the construction of the summary is expensive: first a uniform multidimensional histogram must be computed (with an exponential intermediate memory complexity and low response time), and then the histogram is compressed with the wavelet transform.

Yan et al. [67] improves the wavelet technique by proposing a solution that decreases the intermediate memory required for query answering. This improves the query performance, but the method still suffers from the high summary construction cost of wavelet techniques. Moreover, our solution is based on the kernel additions. Kernels additions outperform histograms in terms of accuracy of the estimated density for both low- and high dimensional data [27, 26]. Finally, we do not need to decompress the density, which is important for fast query answering.

[35] proposes a greedy algorithm which computes sub-optimal wavelets. The greedy algorithm has linear intermediate memory complexity and loglinear time complexity.

2.3 Preliminaries

Grid is the fundamental concept in the AD-Tree data structure. Essentially, grid G is a set of grid points equidistantly spaced in d-dimensional space along each dimension. The number of grid points in grid G is determined by the *granularity* of the grid $\mathsf{S} = ([\mathsf{S}]_1, \ldots, [\mathsf{S}]_d)$ ($[\mathsf{S}]_i$ are positive integers), while the location of grid G is determined by frame $\mathsf{F} = (P, Q)$ (P and Q are real numbers), the lower and upper boundaries of the grid. We conveniently use letters J and λ_J to denote individual grid point in grid G . Multidimensional index $J = ([J]_1, \ldots, [J]_d)$ with $0 \leq [J]_i \leq [\mathsf{S}]_i$, $i = 1, \ldots, d$ refers to the grid point with integers, while $\lambda_J = ([\lambda_J]_1, \ldots, [\lambda_J]_d)$ with $[\lambda_J]_i = [P]_i + [J]_i \cdot \frac{[Q]_i - [P]_i}{[\mathsf{S}]_i}$ refers to the grid point with real values. Given grid point J (λ_J) we refer to the *next* grid point along all dimensions with $J + \mathbb{I} = ([J]_1, \ldots, [J]_d) + (1, \ldots, 1)$ ($\lambda_{J+\mathbb{I}}$), while $J + \mathbb{I}(i) = ([J]_1, \ldots, [J]_d) + (\underbrace{0, 0, \ldots, 0}_{i-1}, 1, \underbrace{0, \ldots, 0, 0}_{d-i})$ denotes to the next grid point along the ith dimension ($\lambda_{J+\mathbb{I}(i)}$). A cell of a grid consists of 2^d grid points of the grid: $\mathsf{C}_J = \{\lambda_{J'} : [J']_i \in \{[J]_i, [J]_i + 1\}, i = 1, \ldots, d\}$ tuples



(a) Grid G.

(b) Kernel Additions Computed At Grid Points

Figure 2.1: Two-Dimensional Grid G.

The AD-Tree is a tree of grids. We use a superscript to specify to which grid the notation refers to. For example, given grids $\mathsf{G}^1, \ldots, \mathsf{G}^l, \ldots, \mathsf{G}^k$, notations S^l, $\mathsf{F}^l = (P^l, Q^l)$, λ^l_J denote the granularity, frame, and the grid point in grid G^l.

Consider two-dimensional grid G in Figure 2.1. $\mathsf{F} = \big((0.2, 0.2), (0.8, 0.6)\big)$ is the frame, $\mathsf{S} = (3, 4)$ is the granularity of the grid. There are $([\mathsf{S}]_1 + 1) \cdot ([\mathsf{S}]_2 + 1) = 20$

grid points in G. The grid points of the grid are:

$$\begin{array}{rcl}
\mathsf{G} = & \begin{array}{cccc}\{\lambda_{(0,4)}, & \lambda_{(1,4)}, & \lambda_{(2,4)}, & \lambda_{(3,4)},\\ \lambda_{(0,3)}, & \lambda_{(1,3)}, & \lambda_{(2,3)}, & \lambda_{(3,3)},\\ \lambda_{(0,2)}, & \lambda_{(1,2)}, & \lambda_{(2,2)}, & \lambda_{(3,2)},\\ \lambda_{(0,1)}, & \lambda_{(1,1)}, & \lambda_{(2,1)}, & \lambda_{(3,1)},\\ \lambda_{(0,0)}, & \lambda_{(1,0)}, & \lambda_{(2,0)}, & \lambda_{(3,0)}\}\end{array} & = \\
& \begin{array}{cccc}\{(0.2,0.6), & (0.4,0.6), & (0.6,0.6), & (0.8,0.6)\\ (0.2,0.5), & (0.4,0.5), & (0.6,0.5), & (0.8,0.5)\\ (0.2,0.4), & (0.4,0.4), & (0.6,0.4), & (0.8,0.4)\\ (0.2,0.3), & (0.4,0.3), & (0.6,0.3), & (0.8,0.3)\\ (0.2,0.2), & (0.4,0.2), & (0.6,0.2), & (0.8,0.2)\}\end{array} &
\end{array}$$

$\mathsf{C}_{(0,0)} = \{\lambda_{(0,0)}, \lambda_{(0,1)}, \lambda_{(1,0)}, \lambda_{(1,1)}\}$ is a cell of the grid.

The kernel estimate [57, 53] for a d-dimensional database containing data points $X^l, l = 1, \ldots, n$, at d-dimensional point x is defined as follows:

$$f(x) = \frac{1}{nh^d} \times \sum_{l=1}^{n} K\Big(\frac{x - X^l}{h}\Big),$$

where K is the kernel function with $K \geq 0$, $\int K = 1$, $K(x) = K(-x)$, and smoothing parameter (window) $h > 0$. Computation of kernel additions requires a scan over the sample of the database and is expensive. Typically, the kernel additions are precomputed at selected grid points and interpolated linearly in between the grid points. This chapter investigates the minimal number and the minimal distribution of grid points for the grid points.

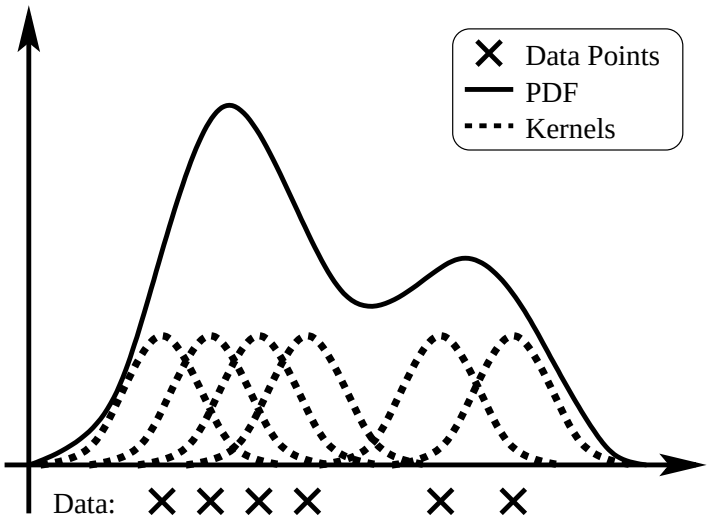


Figure 2.2: Kernel Additions, 1D Case

Notation	Explanation	Example
G,G^l	Grid	$\mathsf{G} = \{\lambda_{(0,0)}, \lambda_{(0,1)}, \lambda_{(0,1)}, \lambda_{(0,1)}\}$ $= \{(0.0, 0.0), (0.1, 0.0),$ $(0.0, 0.2), (0.1, 0.2)\}$
$\mathsf{F} = (P, Q)$	The frame (bounding box) of the grid	$\mathsf{F} = \big((0.0,0.0),(0.1,0.2)\big)$
S	The granularity of the grid	$\mathsf{S} = (2, 2)$
$J, J', \lambda_J^l, \lambda_{J'}^l$	The grid point of the grid	$J = (0, 0), \lambda_J = (0.0, 0.0)$
$\mathbb{I}, \mathbb{I}(i)$	Predefined vectors	$\mathbb{I} = (1, \ldots, 1)$, $\mathbb{I}(i) = (\underbrace{0, 0, \ldots, 0}_{i-1}, 1, \underbrace{0, \ldots, 0, 0}_{d-i})$
C_J	The cell of the grid	$\mathsf{C}_{(0,0)} = \{\lambda_{(0,0)}, \lambda_{(0,1)}, \lambda_{(0,1)}, \lambda_{(0,1)}\}$ $= \{(0.0, 0.0), (0.1, 0.0),$ $(0.0, 0.2), (0.1, 0.2)\}$
$f(x)$	The value of kernel additions at $x \in R$	$f(0.0, 0.0) = 0.0$

Table 2.1: Notations Used in the Chapter

The essence of the kernel method is that each data point increases the chances of having another data point nearby. We draw a symmetric kernel with an area equal to 1 (cf. dashed line, Figure 2.2) around each data point. Adding all kernels yields an estimate of the density (cf. solid line, Figure 2.2). The choice of the kernel does not impact the precision of the estimation and typically the (cheapest in terms of complexity) Epanechnikov's kernel is used:

$$K_{ech}(u) = \begin{cases} a \cdot (1 - u^2) & \text{if } |u| \leq 1, \\ 0 & otherwise. \end{cases}$$

with constant a:

$$a = \frac{\pi^{d/2}(d+2)}{2\Gamma(d/2+1)}$$

and the optimal smoothing parameter:

$$h_{opt} = \Big(\frac{8\Gamma(d/2+1)(d+4)(2\sqrt{\pi})^d}{\pi^{d/2} n}\Big)^{1/(d+4)},$$

where d is the dimensionality of the data, n is the number of points in the dataset, and $\Gamma(z)$ is the gamma function with $\Gamma(k) = (k-1)!$, and $\Gamma(k+1/2) = \frac{1 \cdot 3 \cdot \ldots \cdot (2k-1)}{2^k}\sqrt{\pi}$, where k is an integer.

Table 2.1 summarizes the notations used in the chapter.

2.4 Construction of the AD-Tree

Construction of the AD-Tree with minimal intermediate memory is based on two principals. First, we may not start with a redundant fine grid and remove un-

needed grid points or compress the grid in the refinement step afterwards. The fine partition allocates many grid points uniformly independent whether there are (variations in) data in certain areas of the domain. Computation of the density information on the redundant fine grid becomes prohibitively expensive already for low dimensions (d =4–5) even if the database consists of very simple structures (one dimensional lines or spheres). Second, given a dataset with varying density in an area and a minimal set of grid points that approximate the density tightly (no redundant grid points to meet the error) we need to organize the grid points for storage. Organization of grids points into regular grids saves a lot of memory compared to individual storage of grid points.

The principals suggest an iterative construction of the AD-Tree. The construction starts with a sparse (uniform) grid good enough to "catch" the structures in the data, and adds additional grid points in the areas and only along the dimensions where the grid is too sparse to describe the density linearly. This ensures a monotonic and tight control of the error at each iteration.

Refinement of the grid at each iteration ensures that the grids are not over-split to meet the given error threshold, however introduces very many individual grid points that are expensive to store individually. We organize the individual grid points into grids and optimize their storage.

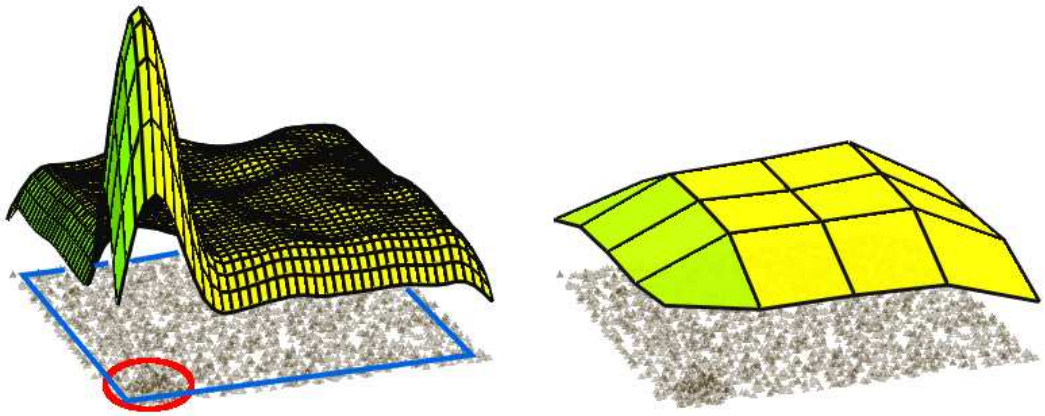
We organize the Section in the following way. In Section 2.4.1 we illustrate the refinement of the AD-Tree that ensures monotonicity minimal over-split, and therefore minimal intermediate memory as iterations increases. In Section 2.4.2 we zoom into one iteration and illustrate the efficient and effective organization of the individual grid points into grids. Finally, we explain in detail the shape error, dimensional split, and the unsplit steps, - the key concepts behind the minimal intermediate memory in Sections 2.4.3, 2.4.4, and 2.4.6 as well as the group step in Section 2.4.5 (optimization of the storage of the individual grid points).

2.4.1 Minimal Intermediate Memory

Figures 2.3 and 2.4 demonstrate the construction of the AD-Tree with minimal intermediate memory on a two dimensional sample dataset. The dataset is illustrated in Figure 2.3(a) and consists of two structures: a plane, which occupies the whole domain, and a sphere, which is located in the left-bottom of the domain. The complexity of the structures differs substantially. The plain exhibits a uniform density in the middle part with a decrease of density towards the outer part of the structure. The sphere is a more complex structure: the density function is not linear in both dimensions.

Construction of the AD-Tree starts with a sparse initial grid. The initial grid is sparse enough so there are no areas with too many grid points for the given error threshold but good enough to "catch" the structure and refine the grids in subsequent steps.

In the first iteration (Figures 2.4(a)–2.4(c)) the outer cells of the grid are too



(a) The Dataset and its Density Computed on a Fine Grid.

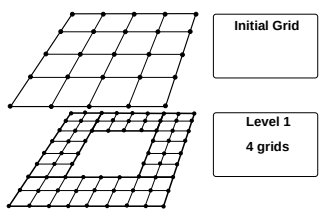
(b) Initial Sparse Grid with the Density Computed at Grid Points.

Figure 2.3: Initialization of the Construction Procedure of the AD-Tree.

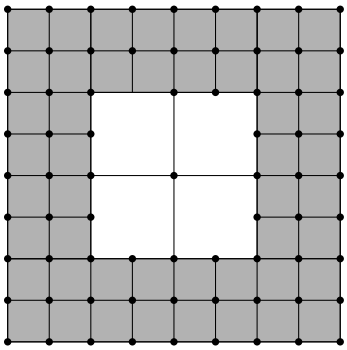
sparse and split along both dimensions. The new partition points are grouped into four regular grids which form a new level of the AD-Tree (Figure 2.4(a)). Clearly, the estimated density become more accurate with a minimal intermediate memory. The refined grid now captures the sphere, however no additional grid points are introduced in the middle of the plane. In contrast, if one started with a very fine partitioning, many points in the middle of the plane would be redundant and be removed resulting in a high intermediate memory.

In the second iteration (Figure 2.4(d)–2.4(f)) the construction does not only effectively identify areas but also detects the dimensions along which the grid points must be added. The grid points are introduced at the corners along the both dimensions (cf. dark-grey cells in Figure 2.4(f)) with some dimensional splits in other outer areas (cf. light-grey cells in Figure 2.4(f)). All the new partition points are grouped into fifteen grids.

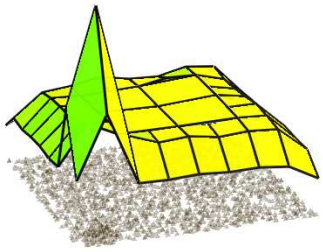
The third iteration completes the construction of the AD-Tree yielding the uniform estimation quality for both structures.



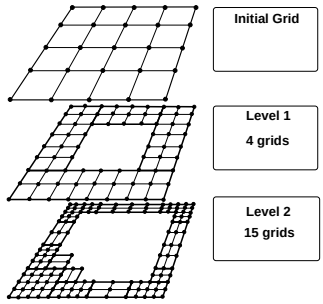
(a) First Iteration: the AD-Tree.



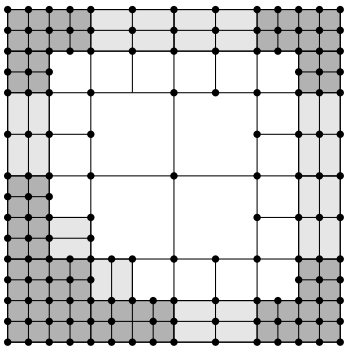
(b) First Iteration: Projected Grid Points.



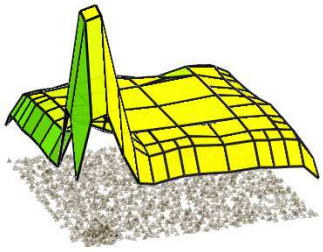
(c) First Iteration: Estimated Density of the Data.



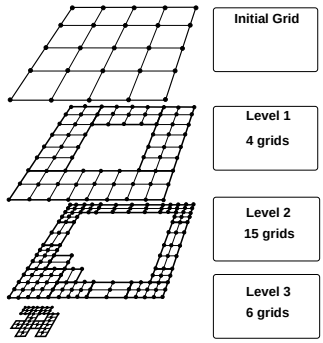
(d) Second Iteration: the AD-Tree.



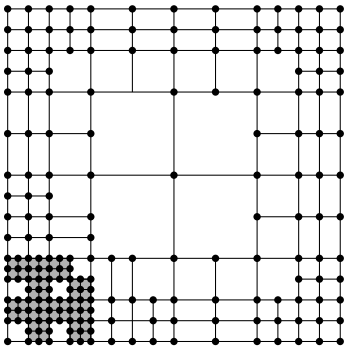
(e) Second Iteration: Projected Grid Points.



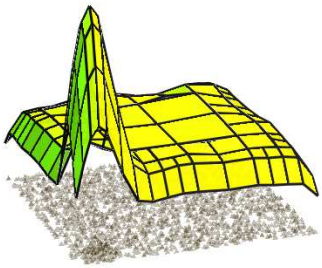
(f) Second Iteration: Estimated Density of the Data.



(g) Third Iteration: the AD-Tree



(h) Third Iteration: Projected Grid Points.

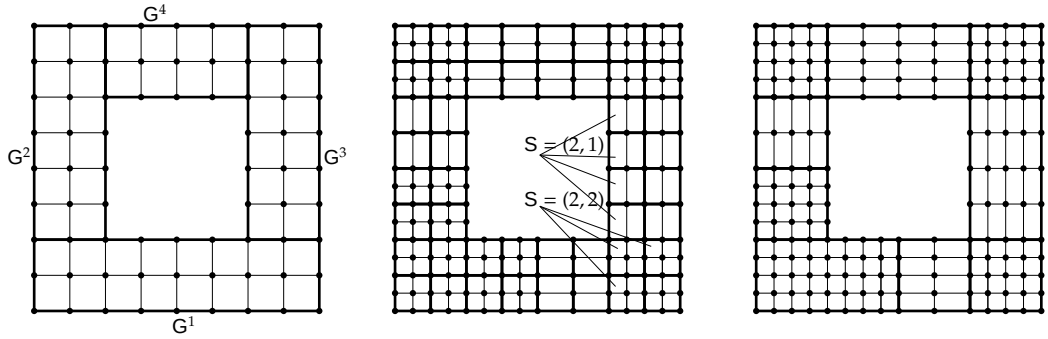


(i) Third Iteration: Estimated Density of the Data.

Figure 2.4: Construction of the AD-Tree. Each Row Illustrates Output of One Iteration.

2.4.2 Optimization of the Storage of the Individual Grid Points and Complete Iteration of the Construction of the AD-Tree

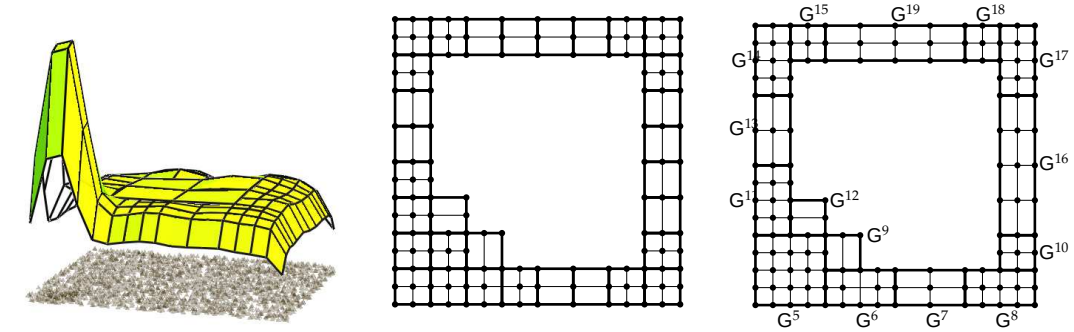
The complete second iteration on the sample dataset in Figure 2.3 is illustrated in Figure 2.5. The iteration starts with the *dimensional split*. The dimensional split inputs the grids of the last level of the AD-Tree (cf. Figure 2.5(b)) checks the *shape error* in the grids and the cells in dimensions of non-linearity of the density function. For each split cell the dimensional split introduces a grid of granularity 2x2. The new grids are organized into larger grids by the *group step* (cf. Figure 2.5(c)). *Kernel additions* compute the density values at the new grids. The density values at the new grid points then are compared against the predicted values. If the prediction is within the given error threshold *unsplit* removes the grid points (cf. Figure 2.5(d)) and the second grouping step (cf. Figure 2.5(e)) re-optimizes the grid for storage.



(a) Input for Second Iteration: 4 Grids of the First Level of the AD-Tree.

(b) Dimensional Split Step: 64 New Grids Produced.

(c) First Group Step: 64 Small Grids are Grouped into 9 Large Grids.



(d) Kernel Additions Step: the Density is Computed on Grid Points of new Grids.

(e) Unsplit Step: 33 Grid Points are Removed and Others are Organized in 31 Grids.

(f) Second Group Step: 31 Small Grids are Grouped into 15 Large Grids.

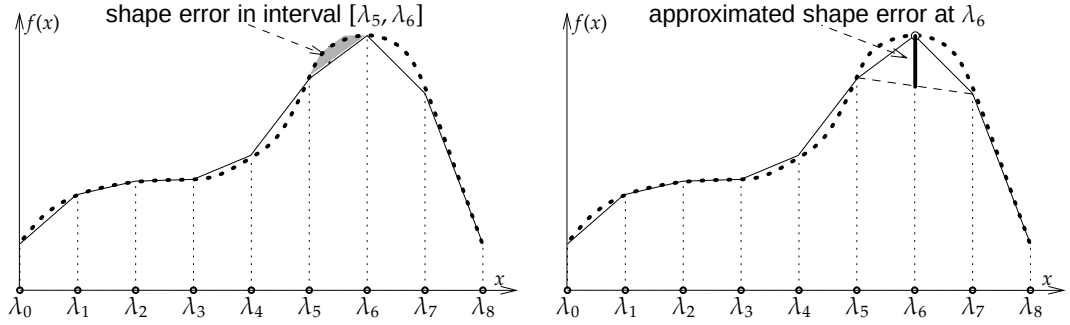
Figure 2.5: A Complete Iteration of Construction Procedure of the AD-Tree.

The dimensional split step and the unsplit step produces grids of small size

(typically 2x2). The group step assigns two small regular grids into the same large regular grid only, if they are adjacent and congruent (their frames and granularity are the same) effectively reducing the number of grids. Since for each grid we need to store the frame and the granularity this allows to substantially reduce the memory usage of the tree. For example In total 64 cells were split and therefore 64 2x2 grids were introduced in the second iteration (cf. Figure 2.5(b)). The grouping step reorganized the grids into nine larger grids. Similarly, 15 larger grids were obtained by the group after the unsplit step.

2.4.3 The Shape Error

The shape error ensures construction of the AD-Tree with minimal intermediate memory. The Shape error identifies and splits cells along the dimensions where the density function is non-linear. This enables minimal intermediate memory usage, since more grid points are added only if the estimation error is above the threshold. In contrast, other methods start with an over-estimate of the density on a very fine grid, and therefore very high intermediate memory. As the number of dimensions increases this becomes prohibitively expensive.



(a) Shape Error in Interval Between Grid Points λ_5 and λ_6.

(b) Approximated Shape Error at Grid Point λ_6.

Figure 2.6: Shape Error and Approximated Shape Error.

The shape error measures the difference between the linear approximation of the density on the grid and the kernel additions (cf. the shaded area in Figure 2.6(a)). The following formalizes shape error:

Definition 2.1. *[Shape Error].*

Let G *be a one-dimensional regular grid and let* λ_J *and* $\lambda_{J'}$ *be two grid points of grid* G *such, that* $\lambda_{J'} > \lambda_J$*. Then, the shape error in the interval between grid points* λ_J *and* $\lambda_{J'}$ *is:*

$$SE(\lambda_J, \lambda_{J'}) = \max_{x \in [\lambda_J, \lambda_{J'}]} \left| f(x) - LI^{f(\lambda_J)}_{f(\lambda_{J'})}(x) \right|,$$

where $f(x)$ is the value of the density at x and $LI^{f(\lambda_J)}_{f(\lambda_{J'})}(x)$ is the linear interpolation between $(\lambda_J, f(\lambda_J))$ and $(\lambda_{J'}, f(\lambda_{J'}))$ at $x \in [\lambda_J, \lambda_{J'}]$:

$$LI^{f(\lambda_J)}_{f(\lambda_{J'})}(x) = f(\lambda_J) + \frac{f(\lambda_{J'}) - f(\lambda_J)}{\lambda_{J'} - \lambda_J}(x - \lambda_J)$$

The shape error is the difference between the linear interpolation and kernel additions. Computation of the shape error is prohibitively expensive since it requires to scan the database in order to compute the kernel additions in the interval. We approximate the shape error using only the density values on the grid points. Given interval $[\lambda_J, \lambda_{J'}]$ and a middle point λ_J in the interval the approximated shape error is the difference between the linear approximation through $\lambda_{J'}, \lambda_J, \lambda_{J''}$ and linear approximation through $\lambda'_J, \lambda_{J''}$. Figure 2.6(b) illustrates the approximated shape error. In general high dimensional case we define the approximated shape error at each grid point of a grid.

We define the approximated shape error for a grid point λ_J in a grid G. We restrict the positioning of the three grid points J, J', J'' so they are consequent grid points in a grid and are always on a line (direction i) parallel to one of the axis of the coordinate system. Below we formalize the definition.

Definition 2.2. *[Approximated Shape Error].*

Let G be a grid and let $\lambda_J \in \mathsf{G}$ be a grid point. Let $f(\lambda_J)$ be the value of the density at grid point λ_J. Then the approximated shape error in dimension i at grid point λ_J is

$$ASE(\mathsf{G}, \lambda_J, i) = \left| LI^{f(\lambda_{J-\mathbb{I}(i)})}_{f(\lambda_{J+\mathbb{I}(i)})}(\lambda_J) - f(\lambda_J) \right| = \left| \frac{f(\lambda_{J-\mathbb{I}(i)}) + f(\lambda_{J+\mathbb{I}(i)})}{2} - f(\lambda_J) \right|,$$

where $\mathbb{I}(i)$ is a d-dimensional vector of 0s everywhere except 1 at position i:

$$\mathbb{I}(i) = (\underbrace{0, 0, \ldots, 0}_{i-1}, 1, \underbrace{0, \ldots, 0, 0}_{d-i})$$

and $\lambda_{J-\mathbb{I}(i)}$ and $\lambda_{J+\mathbb{I}(i)}$ are the grid points of grid G adjacent to λ_J in dimension i.

Example 2.1. *[Approximated Shape Error].*

Consider Figure 2.7. dimension $i = 2$ and grid point $\lambda^0_{1,1}$ of the two-dimensional regular grid G^0. The following points are adjacent to $\lambda^0_{1,1}$:

$$\lambda^0_{(1,1)+\mathbb{I}(2)} = \lambda^0_{(1,1)+(0,1)} = \lambda^0_{1,2}$$

and

$$\lambda^0_{(1,1)-\mathbb{I}(2)} = \lambda^0_{(1,1)-(0,1)} = \lambda^0_{1,0}$$

The approximated shape error along dimension $i = 2$ at grid point $\lambda^0_{1,1}$ is:

$$ASE(\mathsf{G}^0, \lambda^0_{1,1}, 2) = \left| LI^{f(\lambda^0_{1,0})}_{f(\lambda^0_{1,2})}(\lambda^0_{1,1}) - f(\lambda^0_{1,1}) \right| = \left| \frac{f(\lambda^0_{1,0}) + f(\lambda^0_{1,2})}{2} - f(\lambda^0_{1,1}) \right|$$

Let us assume the following values of density at grid points $\lambda^0_{1,0}$, $\lambda^0_{1,1}$ *and* $\lambda^0_{1,2}$*:*

$$f(\lambda^0_{1,0}) = 0.09,$$
$$f(\lambda^0_{1,1}) = 0.15,$$
$$f(\lambda^0_{1,2}) = 0.1,$$

Substituting these values gives the following approximated shape error:

$$ASE(\mathsf{G}^0, \lambda^0_{1,1}, 2) = \left|\frac{0.09 + 1.0}{2} - 0.15\right| = 0.055$$

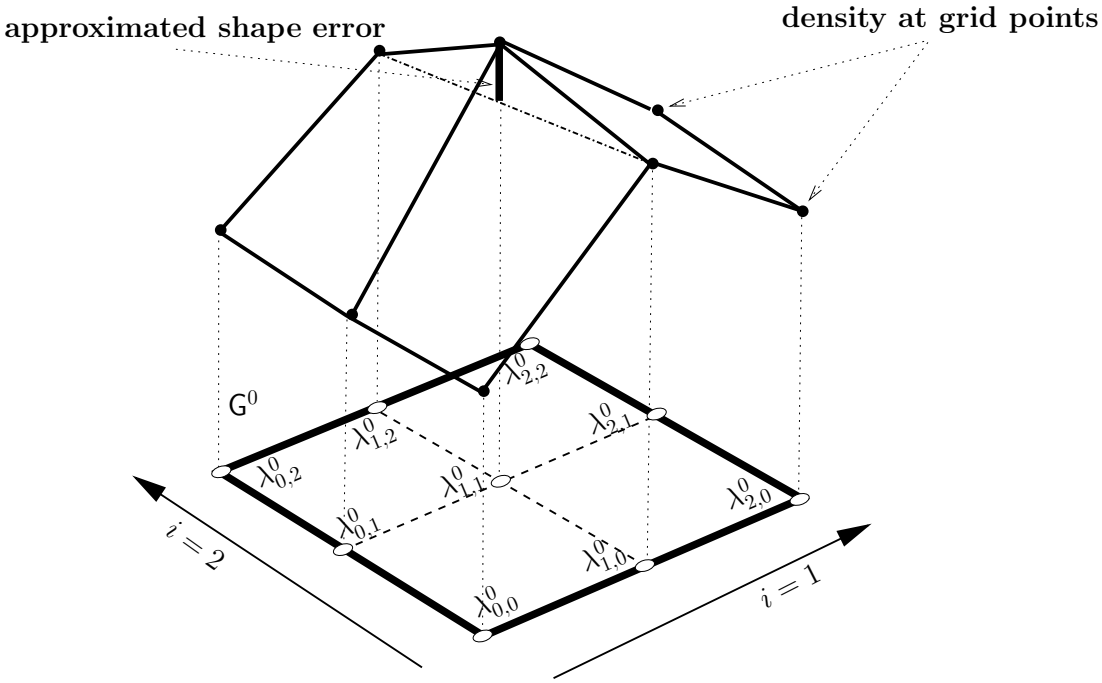


Figure 2.7: Approximated Shape Error along Dimension $i = 2$ at Grid Point $\lambda^0_{1,1}$ of Two-dimensional Grid G^0.

2.4.4 Dimensional Split

The dimensional split ensures that cells with high shape error are split only along the dimensions of non-linearity of the density. For a given grid, the dimensional split checks all cells and all directions along which the shape error is too high and splits the cells (introduces individual grids for each split cell) accordingly.

Consider cell $\mathsf{C}_{1,1,1}$ of the 3-dimensional regular grid ($d = 3$), in Figure 2.8. Let us assume that the density exhibits non-linear behavior along two the 1st and 3rd dimensions (x and z axis) but is linear along the 2nd dimension (y axis), i.e. the approximated shape error exceeds the given threshold at one (or more)

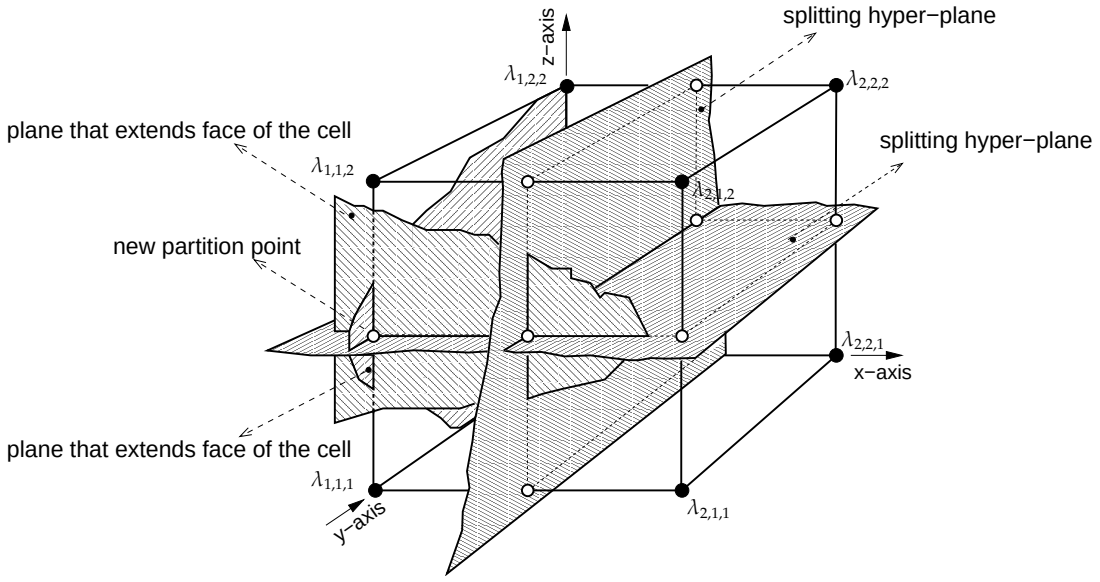


Figure 2.8: Idea of Dimensional Split: Intersection of Splitting Hyper-Planes with Faces of 3-dimensional Cell $\mathsf{C}_{1,1,1}$.

grid point of the cell in dimension 1 and 3. The split of the cell is a new (finer) grid obtained with the splitting hyper planes. In figure 2.8 one splitting hyper-plane is perpendicular to x-axis and another splitting hyper-plane to z-axis. The intersection points of the splitting hyper-planes and faces of the cell yields new grid points (cf. empty circles in Figure 2.8). These grid points are organized in a new grid. The following definition 2.3 formalizes the output of the dimensional split.

Definition 2.3. *[Dimensional Split of a Cell].*

Let $\mathsf{C}_J \subseteq \mathsf{G}$ *be a cell of a d-dimensional grid. Then grid* $\mathrm{ds}(\mathsf{C}_J)$ *is a dimensional split of cell* C_J *iff*

1. *Frame* F *of* $\mathrm{ds}(\mathsf{C}_J)$ *is the bounds of the cell:*

$$\mathsf{F} = (\lambda_J, \lambda_{J+\mathbb{I}}),$$

where $\mathbb{I} = (1, \ldots, 1)$ *and* $\lambda_J, \lambda_{J+\mathbb{I}} \in \mathsf{C}_J$.

2. *Let the granularity of* $\mathrm{ds}(\mathsf{C}_J)$ *be denoted by* $\mathsf{S} = ([\mathsf{S}]_1, \ldots, [\mathsf{S}]_d)$. *Then*

$$[\mathsf{S}]_i = \begin{cases} 2 & \text{if } \max\limits_{\lambda_{J'} \in \mathsf{C}_J} \mathit{ASE}(\mathsf{G}, \lambda_{J'}, i) > \varepsilon \\ 1 & \text{otherwise.} \end{cases}$$

The first condition of Definition 2.3 requires that grid $\mathrm{ds}(\mathsf{C}_J)$ has vertexes of its frame at grid points of the target cell C_J. For example, the frame of grid

$\mathrm{ds}(\mathsf{C}_J)$ in Figure 2.8 is $\mathsf{F} = (\lambda_{1,1,1}, \lambda_{2,2,2})$. Vertexes of F are grid points of the target cell. The second condition of Definition 2.3 defines the granularity of grid $\mathrm{ds}(\mathsf{C}_J)$. Depending on the approximated shape error at the grid points of the target cell granularity of $\mathrm{ds}(\mathsf{C}_J)$ can be either two or one along each dimension. For example, in Figure 2.8 approximated shape error at the grid points of the cell exceeds the given threshold in dimensions 1 and 3. Hence, granularity of output grid $\mathrm{ds}(\mathsf{C}_J)$ is two in dimensions 1 and 3, and one - in dimension 2, i.e. $\mathsf{S} = (2, 1, 2)$.

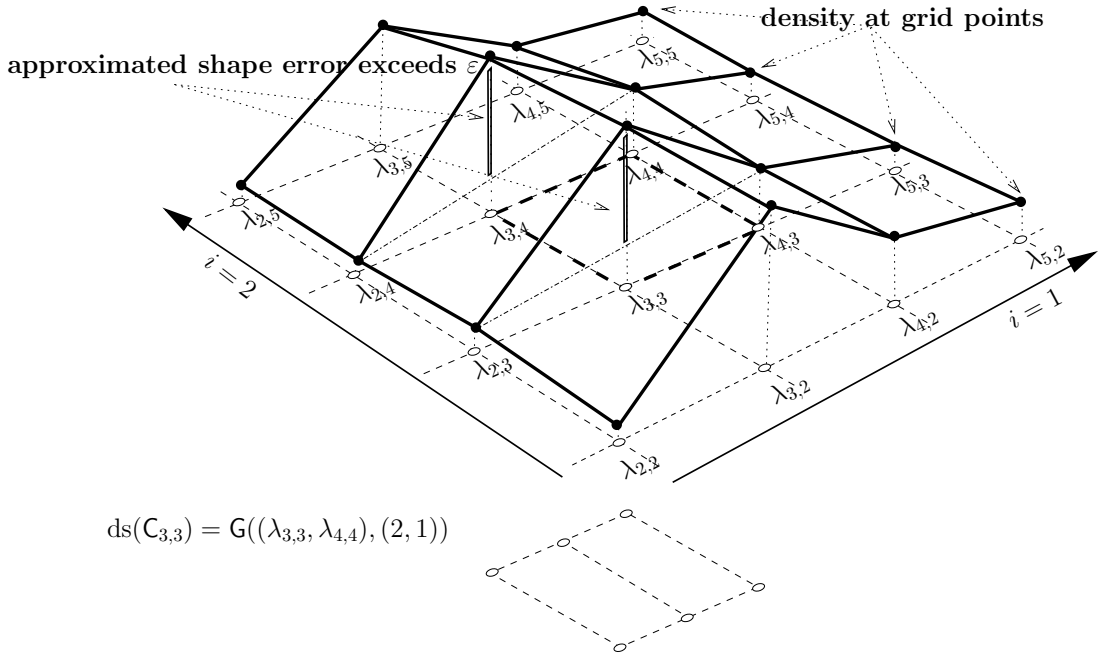


Figure 2.9: Dimensional Split of 2-dimensional Cell $\mathsf{C}_{3,3}$.

Example 2.2. *[Dimensional Split of a Two-dimensional Cell].*

Consider the 2-dimensional grid G *on Figure 2.9*

The dimensional split of cell $\mathsf{C}_{3,3} = \{\lambda_{3,3}, \lambda_{4,3}, \lambda_{3,4}, \lambda_{4,4}\} \in \mathsf{G}$ *is grid* $\mathrm{ds}(\mathsf{C}_{3,3})$ *with frame* $\mathsf{F} = (\lambda_{3,3}, \lambda_{4,4})$*. The granularity of the grid depends on the approximated shape error and is computed as follows. Let us assume* $\varepsilon = 0.05$ *and the density values for the grid points in Figure 2.9 are:*

$$\begin{array}{llllllllllll}
f(\lambda_{2,2}) & = & 0.02, f(\lambda_{2,3}) & = & 0.02, & f(\lambda_{2,4}) & = & 0.022, & f(\lambda_{2,5}) & = & 0.019, \\
f(\lambda_{3,2}) & = & 0.18, f(\lambda_{3,3}) & = & 0.2, & f(\lambda_{3,4}) & = & 0.2, & f(\lambda_{3,5}) & = & 0.21, \\
f(\lambda_{4,2}) & = & 0.07, f(\lambda_{4,3}) & = & 0.08, & f(\lambda_{4,4}) & = & 0.078, & f(\lambda_{4,5}) & = & 0.07, \\
f(\lambda_{5,2}) & = & 0.04, f(\lambda_{5,3}) & = & 0.04, & f(\lambda_{5,4}) & = & 0.04, & f(\lambda_{5,5}) & = & 0.045.
\end{array}$$

Then the approximated shape error along dimension $i = 1$ *at grid points* $\lambda_{3,3}$,

$\lambda_{4,3}$, $\lambda_{3,4}$ and $\lambda_{4,4}$ of cell $\mathsf{C}_{3,3}$ are:

$$\begin{aligned}
\mathit{ASE}(\mathsf{G}, \lambda_{3,3}, 1) &= \left|\tfrac{f(\lambda_{2,3})+f(\lambda_{4,3})}{2} - f(\lambda_{3,3})\right| = \left|\tfrac{0.02+0.08}{2} - 0.2\right| = 0.15,\\
\mathit{ASE}(\mathsf{G}, \lambda_{4,3}, 1) &= \left|\tfrac{f(\lambda_{3,3})+f(\lambda_{5,3})}{2} - f(\lambda_{4,3})\right| = \left|\tfrac{0.2+0.04}{2} - 0.08\right| = 0.04,\\
\mathit{ASE}(\mathsf{G}, \lambda_{3,4}, 1) &= \left|\tfrac{f(\lambda_{2,4})+f(\lambda_{4,4})}{2} - f(\lambda_{3,4})\right| = \left|\tfrac{0.022+0.078}{2} - 0.2\right| = 0.15,\\
\mathit{ASE}(\mathsf{G}, \lambda_{4,4}, 1) &= \left|\tfrac{f(\lambda_{3,4})+f(\lambda_{5,4})}{2} - f(\lambda_{4,4})\right| = \left|\tfrac{0.2+0.04}{2} - 0.078\right| = 0.042.
\end{aligned}$$

The maximum approximated shape error along dimension $i = 1$ is 0.15 *and exceeds given threshold $\varepsilon = 0.05$. Therefore, the granularity of* $\mathrm{ds}(\mathsf{C}_J)$ *along dimension $i = 1$ is $[\mathsf{S}]_2 = 2$. Similarly, the granularity along dimension $i = 2$ is $[\mathsf{S}]_2 = 1$.*

Dimensional split of a cell may produce a grid of the same size as the cell. If the approximated shape error at all grid point and in all dimension i of a cell does not exceed the given threshold ε, then the granularity of the G is a unit vector and $\mathrm{ds}(\mathsf{C}_J) = \mathsf{C}_J$. In this case, we extend the definition so the dimensional split of the cell is the empty set.

We define the dimensional split of a grid as the set of the dimensional splits of all cells of the grid.

Definition 2.4. *[Dimensional Split of a Grid].*

Let G be a d-dimensional grid. Then

$$\mathrm{ds}(\mathsf{G}) = \{\mathrm{ds}(\mathsf{C}) : \mathsf{C} \in \mathsf{G} \text{ and } \mathrm{ds}(\mathsf{C}) \neq \emptyset\}$$

is the dimensional split of grid G.

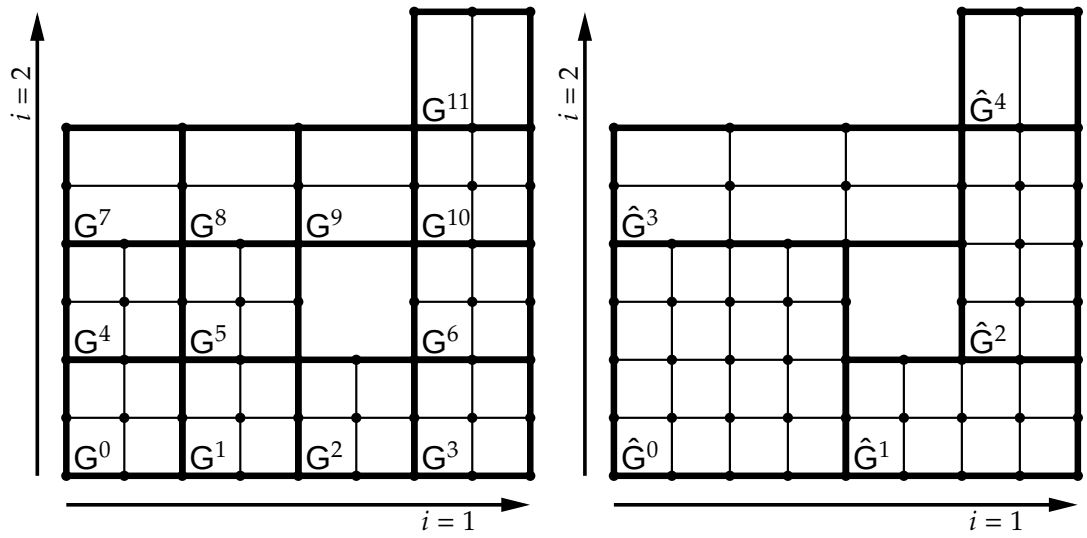
2.4.5 The Group Step

The dimensional split and unsplit steps introduce individual small grids. The group step organize the individual small grids into larger grids saving memory and speeding up both the kernel additions of the grids points as well as querying the AD-Tree.

Figure 2.10 illustrates the group step. There are twelve grids of small size introduced by the dimensional split. The group step reorganizes the grids into five large grids. Reorganization of the grids is an NP hard problem. Instead, we develop a solution which in linear time reduces the total number of grids by maximizing volume of frames of the grid. In this section we formalize the group step and give the idea of our solution. The detail algorithm for the group is given in Section 2.8.

Definition 2.5. *[Grouping].*

Let $\{\mathsf{G}^0, \mathsf{G}^1, \ldots, \mathsf{G}^k\}$ be a set of (so called small) grids. The goal of grouping is to find the minimal number of such d-dimensional (so called maximal) grids $\{\hat{\mathsf{G}}^1, \ldots, \hat{\mathsf{G}}^l\}$ that:



(a) Input (b) Output

Figure 2.10: Group Step Minimizes Number of Grids from Twelve to Five.

1. $\hat{\mathsf{G}}^m$ *is a union of* G^l *of the same granularity.*

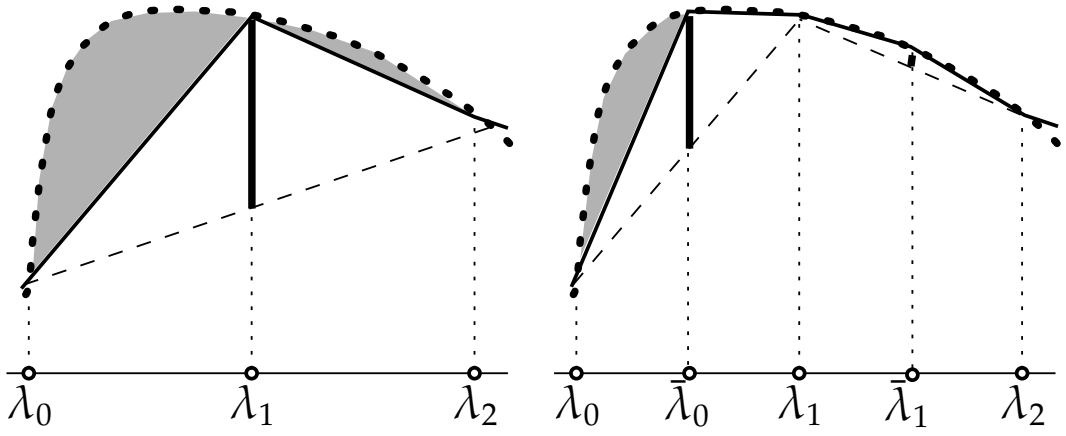
2. *Frames of any two grids* $\hat{\mathsf{G}}^{l_1}$ *and* $\hat{\mathsf{G}}^{l_2}$ *do not overlap.*

The idea of our grouping step is the following. At each iteration of the grouping algorithm we select the grid that (i) is not yet processed and (ii) closest to the origin according to all dimensions (smallest along the 1st, then the 2nd, etc.) This is the starting grid of the maximal grid. Then we expand that frame of the maximal grid along axes so two conditions are satisfied. First, all small grids which are enclosed by the expanded frame are of the same granularity. Second, the expanded frame does not contain gaps between the enclosed grids.

2.4.6 Unsplit Step

If the (true) shape error is above the given threshold we need to split cells to meet the error. Addition of grid points based on the (true) shape error is required and *necessary* and always reduces the error. The dimensional split is based on the approximate shape error may oversplit the cells especially at the boundaries of the areas of linearity of the density. Figure 2.11 illustrates the issue. The approximate shape error at λ_1 is above the error and both intervals $[\lambda_0, \lambda_1]$ and $[\lambda_1, \lambda_2$ are split. However, the linear approximation of the density is already good enough in interval $[\lambda_0, \lambda_2]$. The unsplit step removes such unnecessary points and eliminates grid points that overestimate the density.

Conceptually, the unsplit step operates on the grids introduced by the dimensional split and checks whether some or all of them overestimate density



(a) Input (b) Output

Figure 2.11: Oversplit.

function in one or more dimensions. We say that grid point λ overestimates density function along dimension i if the approximated shape error at grid point λ along dimension i is below given threshold ε. Unsplit step removes all grid points which overestimate density and are not required to support the grid structure. Below we define the unsplit for a special case. We assume that there is only one cell split by the dimensional split.

Definition 2.6. *[Unsplit of a Cell].*

Let C *be a cell and* $\mathrm{ds}(\mathsf{C})$ *be a dimensional split of cell* C*. Grid* $\mathrm{unsplit}(\mathsf{C}, \mathrm{ds}(\mathsf{C}))$ *is the unsplit of* $\mathrm{ds}(\mathsf{C})$ *iff*

1. *The frames of grids* $\mathrm{ds}(\mathsf{C})$ *and* $\mathrm{unsplit}(\mathsf{C}, \mathrm{ds}(\mathsf{C}))$ *are the same*
2. *Let* $\mathsf{S} = ([\mathsf{S}]_1, \ldots, [\mathsf{S}]_d)$ *be the granularity of* $\mathrm{unsplit}(\mathsf{C}, \mathrm{ds}(\mathsf{C}))$*. Then*

$$[\mathsf{S}]_i = \begin{cases} 2 & \textit{if } \mathit{ASE}(\mathrm{ds}(\mathsf{C}), \lambda_{J(i)}, i) > \varepsilon \textit{ for some } \lambda_{J(i)} \in \mathsf{C} \\ 1 & \textit{otherwise,} \end{cases} \tag{2.1}$$

where $J(i) = ([J]_1, \ldots, [J]_{i-1}, 1, [J]_{i+1}, \ldots, [J]_d)$*,* $[J]_j \in \{1, 3\}, j = 1, \ldots, i-1, i+1, \ldots, d$*.*

The unsplit inputs a split cell $\mathrm{ds}(\mathsf{C})$ and checks whether all middle points can be removed along dimension i (cf. Equation 2.1). Similarly to the dimensional split unsplit of a cell is a d-dimensional grid with the granularity one or two in each dimension i. Grid $\mathrm{unsplit}(\mathsf{C}, \mathrm{ds}(\mathsf{C}))$ is of granularity 2 in dimension i only, if cell C is split in that dimension i and at all grid points introduced by this split the approximated shape error in dimension i does not exceed given threshold ε.

Example 2.3. *[Unsplit of a 2-dimensional Cell].*

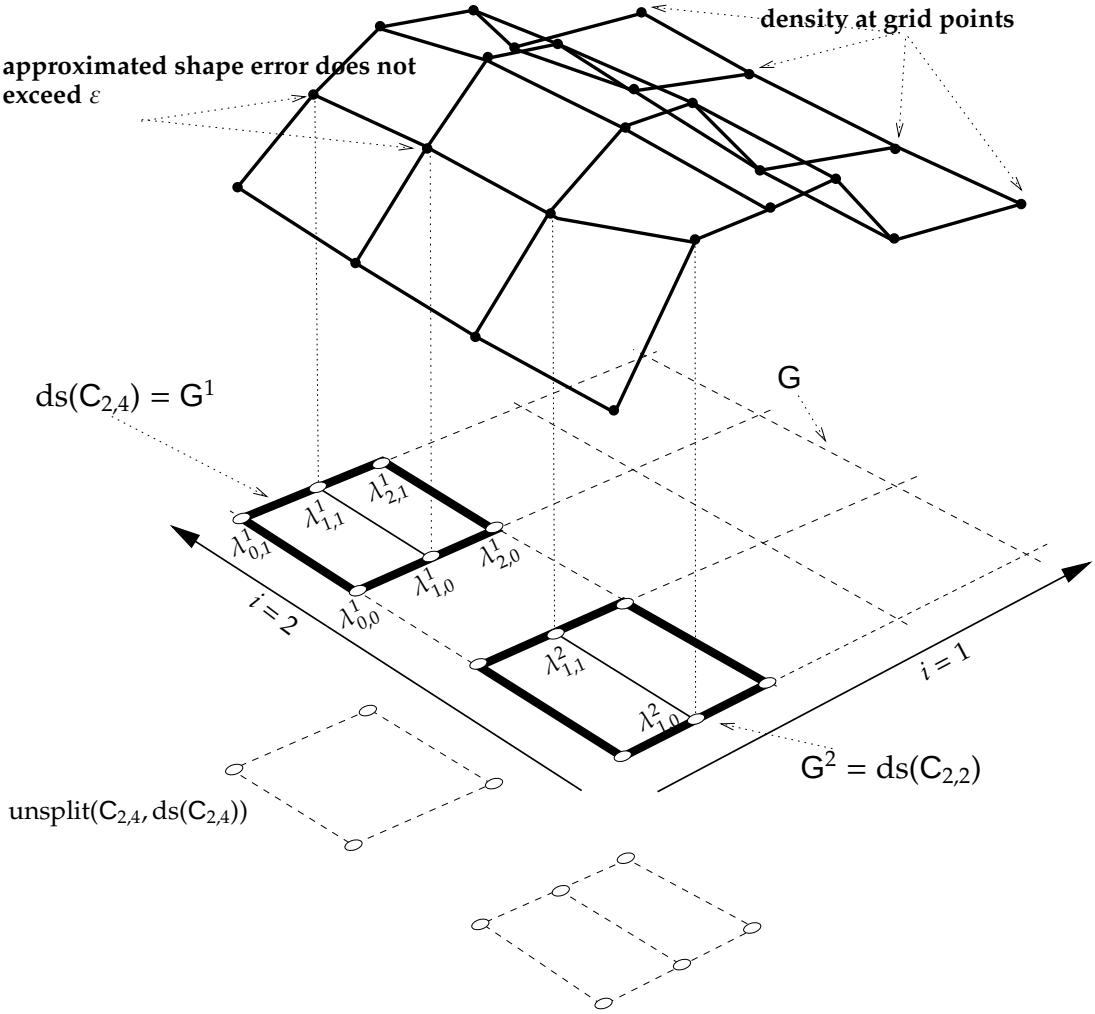


Figure 2.12: Unsplit of 2-dimensional Cells $C_{2,4}$ and $C_{2,2}$.

Let us consider cell $\mathsf{C}_{2,4}$ *of* 2*-dimensional grid* G *in Figure 2.12. Let split(*$\mathrm{ds}(\mathsf{C})$*) be the dimensional split of this cell in dimension* $i = 1$ *with granularity* $\mathsf{S} = (2, 1)$*. Let as assume* $\varepsilon = 0.05$ *and the following density values for grid points of grid* G*:*

$$f(\lambda^1_{0,0}) = 0.022, \quad f(\lambda^1_{1,0}) = 0.031, \quad f(\lambda^1_{2,0}) = 0.04,$$
$$f(\lambda^1_{0,1}) = 0.019, \quad f(\lambda^1_{1,1}) = 0.03, \quad f(\lambda^1_{2,1}) = 0.39$$

Neither $\lambda_{1,0}$ *nor* $\lambda_{1,1}$ *are required to meet the threshold. Indeed, let us compute the approximated shape errors for the middle points:*

$$\begin{aligned} \mathit{ASE}(\mathsf{G}^1, \lambda^1_{1,0}, 1) &= \left| \tfrac{f(\lambda^1_{0,0}) + f(\lambda^1_{2,0})}{2} - f(\lambda_{1,0}) \right| = \left| \tfrac{0.022 + 0.04}{2} - 0.031 \right| = 0.0, \\ \mathit{ASE}(\mathsf{G}^1, \lambda^1_{2,1}, 1) &= \left| \tfrac{f(\lambda^1_{0,1}) + f(\lambda^1_{2,1})}{2} - f(\lambda^1_{1,1}) \right| = \left| \tfrac{0.019 + 0.039}{2} - 0.03 \right| = 0.001. \end{aligned}$$

The approximated shape error at both grid points does not exceed given threshold $\varepsilon = 0.05$ *and, hence granularity of grid* $\mathrm{unsplit}(\mathsf{C}_{2,2}, \mathrm{ds}(\mathsf{C}_{2,2}))$ *in dimension* $i = 1$ *is one.*

In contrast to the unsplit of cell $\mathsf{C}_{2,4}$*, the unsplit of cell* $\mathsf{C}_{2,2}$ *is the same* $\mathrm{ds}(\mathsf{C}_{2,2})$*. The approximated shape error in dimension* $i = 1$ *is high at grid point* $\lambda^2_{1,0}$ *and low at grid point* $\lambda^2_{1,1}$*.*

2.5 Data Structure of the AD-Tree

We carefully engineer a data structure for the AD-Tree for efficient storage of the grids and the tree structure. We define the compact representation of the grids in the AD-Tree. The grids in the AD-Tree are represented with two digits for the frame of the grid and a bit array of d bits for the granularity vector. The straight forward cost to store the frame is $2 \cdot d$ and d digits for the granularity vector of a grid. The compact representation reduces the storage cost of the frame by a factor of d and storage cost of the granularity vector by the length of a digit.

Figure 2.13 illustrates the compact representation. All grid points in G^3 are assigned with the ids of row-major ordering [36]. The row-major ordering ids then are used to denote the frames of the grids. For example, grid G^3 is a child of the initial grid G^0. $\mathsf{F}^3 = (\lambda^0_{3,1}, \lambda^0_{4,4})$ is the frame of G^3 and is encoded with row-major order ids $(8, 24)$. In the compact representation the granularity of a grid is a bit array of d bits. The bit is full in dimension i iff the cells of the grid split its parent in dimension i and zero otherwise. For example the bit array of G^3 is $(1, 1)$ (two full bits), since grid G^3 splits G^0 along both dimensions.

The following definition formalizes compact representation of maximal grids.

Definition 2.7. *[Compact Representation of a Grid].*

Let G^p *be the parent grid and* G^c *be the child grid. Then, the compact representation of* G^c *is a vector* $(\mathsf{l}, \mathsf{u}, (\mathsf{B}_1, \ldots, \mathsf{B}_d))$ *such that*

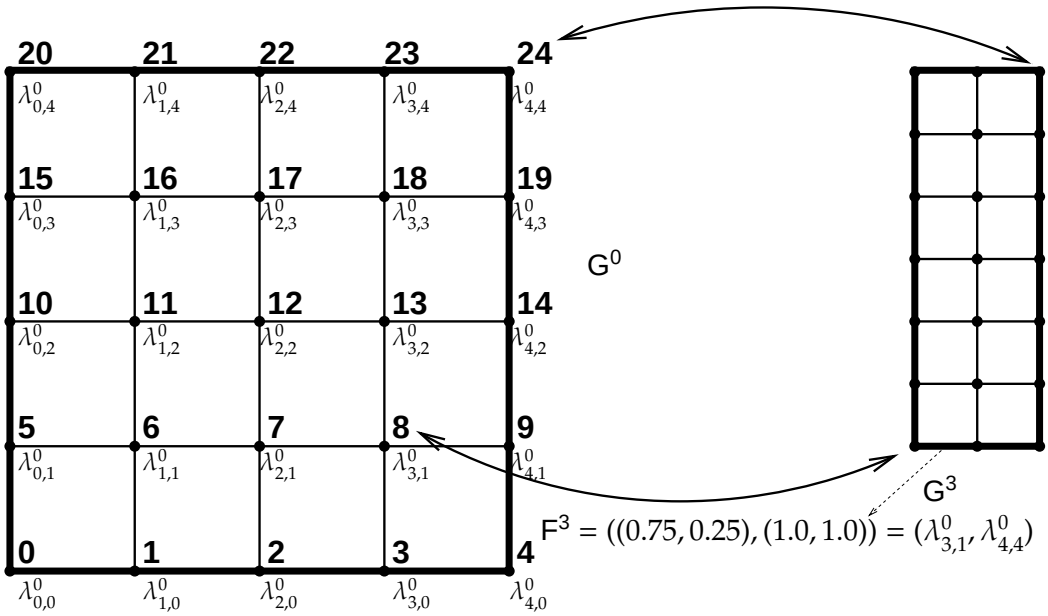


Figure 2.13: Compact Representation of a Grid.

1. *Let* $\lambda^p_J, \lambda^p_{J'} \in \mathsf{G}^p$ *be the frame of* G^c*, and* $\mathsf{S}^p = ([\mathsf{S}]^p_1, \ldots, [\mathsf{S}]^p_d)$ *be the granularity of* G^p*. Then the row-major ordering ids of frame* F^c *are computed as follows:*

$$\mathsf{l} = \sum_{i=1}^{d} \Big(\prod_{r=1}^{i} [\mathsf{S}]^p_r \Big) [J]_i,$$

$$\mathsf{u} = \sum_{i=1}^{d} \Big(\prod_{r=1}^{i} [\mathsf{S}]^p_r \Big) [J']_i.$$

2. $\mathsf{B}_i = 1(0)$ *if all cells of grid* G^p *are (not) split along dimension* i*,* $1 \leq i \leq d$*.*

Figure 2.14 shows the storage of the AD-Tree (cf. Figure 2.4) in linear memory. The AD-Tree consists of three levels and twenty six grids. Each node in Figure 2.14 represents one of these grids with the following parent-child relationship: a node representing grid G^p is a parent node of node which represents grid G^c only, if grid G^c is a product of the dimensional split, unsplit and group steps of grid G^p. In the Figure 2.14, we use white blocks to denote individual nodes. The number in black within each white block corresponds to index of a grid represented by the node and two vectors is compact representation of this grid (except for the root node). We arrange nodes according to their parent-child relationship so, that children of the same parent node form a sequence. In Figure 2.14, white blocks enclosed by a black block are children of the same parent node and form a sequence. Organization of children nodes into one sequence minimizes number

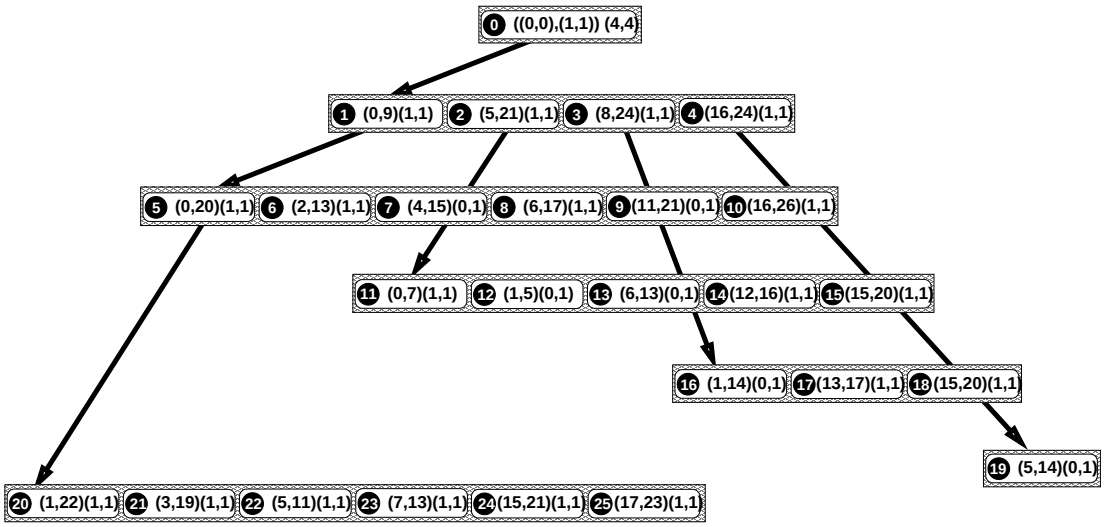


Figure 2.14: Organization of the Internal Memory for the AD-Tree.

of pointers required to reference child nodes: independently to the number of children each node has one pointer to its first child. In Figure 2.14, black arrows denote pointers which reference parent node with its first child. For example, node representing grid G^1 has six children and one pointer which reference its first child node. Other children of a node are accessed by a sequential scan. Root node of the AD-Tree stores frame and granularity of the initial grid, other nodes store compact representation of grids. In a two dimensional case, compact representation of a grid requires three digits that is twice less comparing to the normal representation. In the example, 126 digits are required to store grids normally and 63 digits are used. Each node contains an array which stores density measures computed at grid points of the grid.

2.6 Analytical Investigation

In this section we establish three properties of the AD-Tree. First, we show that the AD-Tree is minimal for a given initial partitioning (cf. Section 2.6.1) and formulate estimation guarantees once a good initial partitioning is given. Second, we define the concept of local dimensionality of a structure in the dataset, and prove that AD-Tree adjusts to the local dimensionality of the structure (cf. Section 2.6.2). Third, we show how to compute a good initial partitioning. More specifically, we show that initial partitioning based on extrema points of the density provides robust approximation of the shape error in the whole domain of the data, and we show how to estimate extrema points of the density with the help of one-dimensional histograms.

2.6.1 Optimality of the AD-Tree for a Given Initial Partitioning

The minimal partitioning for a given density function can be formalized in the following way. Find the smallest number of grids k and the best distribution of the grids $\mathsf{G}^1, \mathsf{G}^2, \ldots, \mathsf{G}^k$ (that can be stored in the AD-Tree) that ensures uniform quality of the estimation:

$$\min_{k} \max_{\mathsf{G}^1,\mathsf{G}^2,\ldots,\mathsf{G}^k} \max_{\lambda_J,\lambda_{J'}\in\mathsf{G}^i} \mathsf{SE}(\lambda_J, \lambda_{J'}) < \varepsilon, \tag{2.2}$$

where $\mathsf{SE}(\lambda_J, \lambda_{J'})$ is the shape error through any two corner points of a cell of a grid.

Computation of the AD-Tree satisfying Equation (2.2) is problematic. A typical approach is to take the derivative of the function that is being minimized or maximized, however the shape error is not differentiable at many points of the domain. It is possible to break down the domain into individual rectangles of differentiability, however, that is very computationally expensive due to multiple computation of kernel additions involved by the shape error. Alternatively, one could pre-compute the kernel additions on a very fine uniform partition, and search for the smallest size and the best distribution of the partition, such that the coordinates of partition points are on the fine partition. However this approach is exponential in terms of the k.

To reduce the search space, we make the following assumption to find the minimal partitioning. We start with a given (sparse) initial partitioning and allow the initial partitioning to be further split into equal-sized hyper-rectangles. The following result shows that the AD-Tree yields a minimal partitioning, provided that the shape error is estimated robustly.

Theorem 2.1. *At the end of the estimation process the number of partition points in the AD-Tree is optimal with respect to the linear initial partition, provided that the SE is approximated robustly and partition points are stored on a grid.*

PROOF: We prove the theorem by a contradiction. Let AD-Tree be the final tree that we derived from the initial grid and estimation error ε. Assume, the partition is not optimal, i.e., there exists a grid G^p in the partition such its child grid G^c of one of its cells C^p is not required, i.e.,

$$\mathsf{SE}(\lambda^c_J, \lambda^c_{J'}) \leq \varepsilon, \tag{2.3}$$

where λ^c_J and $\lambda^c_{J'}$ are adjacent grid points in G^c.

The AD-Tree introduces new partition points only if the new partition points decrease the shape error or are necessary to support the grid structure. Since λ^c_J is a partition point, therefore it decreases the shape error or there exists another partition point $\lambda^c_{J''}$ such that, $\lambda^c_J, \lambda^c_{J''}$ are in the same cell and $\lambda^c_{J''}$ decreases the

shape error:

$$\mathsf{SE}(\lambda^c_J, \lambda^c_{J'}) > \varepsilon \text{ or } (\mathsf{SE}(\lambda^c_J, \lambda^c_{J''}) > \varepsilon \text{ and } \mathsf{SE}(\lambda^c_J, \lambda^c_{J'}) \leq \varepsilon). \tag{2.4}$$

The contradiction proves the theorem. □

2.6.2 Local Dimensionality of the Data

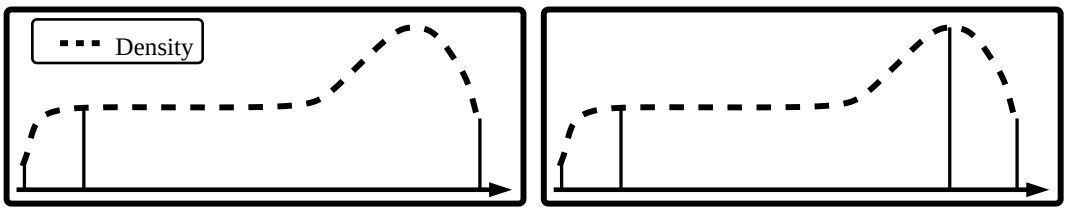
Definition 2.8. *Let $S \subset D$ be a structure of the database such that the density in domain of S is non-linear along $1, \ldots, d'$ and linear along the remaining dimensions $d'+1, \ldots, d$. Then the local dimensionality of the structure is d'.*

Theorem 2.2. *[AD-Tree adjusts to the local dimensionality of the structures]. Let $S \subset D$ be a d'-local dimensionality structure. Let $1, \ldots, d'$ be the dimensions along which the density is non linear; the initial partitioning is chosen so the shape error can be estimated robustly. Then AD-Tree adjusts to the local dimensionality of the structure: the AD-Tree method will split the space along $1, \ldots, d'$ and will not split along the remaining dimensions $d'+1, \ldots, d$.*

PROOF: Proof follows from the Definition 2.3. □

2.6.3 Initial Partitioning Based on the Extrema Points

The initial partitioning should be dense enough in order to approximate the shape error robustly. If the initial partitioning is chosen too sparse, the approximated shape error may be be below the threshold, however the true shape error be above the threshold (cf. the are at the peak in Figure 2.15(a)). The initial partitioning with an additional point at the peak of the density would make the approximation of the shape error robustly (cf. Figure 2.15(a)).



(a) Too Sparse Initial Partitioning (b) Good Initial Partitioning

Figure 2.15: Approximation of the shape error

In the remaining of the section we show that the initial partitioning based on the extrema points of the density guarantees a robust approximation of the shape error. Consider a one-dimensional case. Then the following is true.

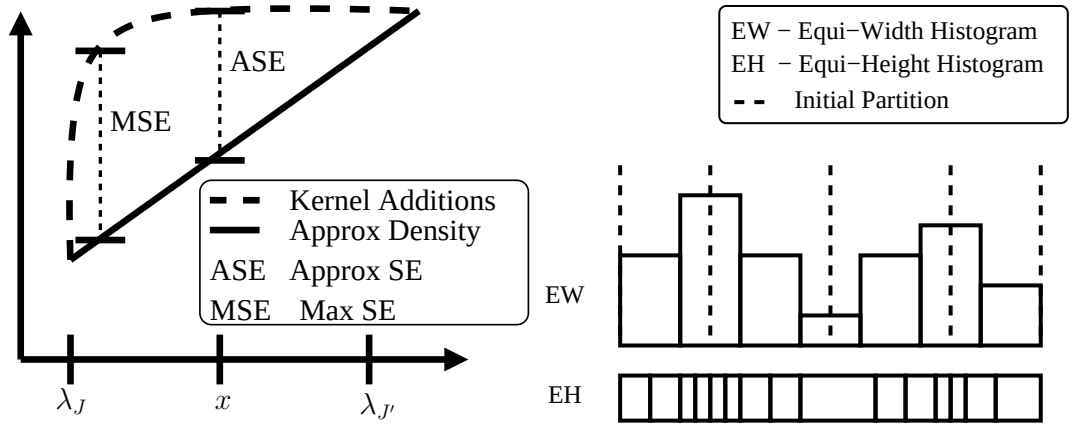
Theorem 2.3. *Let λ_J and $\lambda_{J'}$ be two adjacent partition points at extrema points of the density. Then, on average the approximate shape error approximates the shape error:*

$$\frac{1}{|\lambda_J - \lambda_{J'}|} \int_{x \in [\lambda_J, \lambda_{J'}]} |\mathit{ASE}(\{\lambda_J, \lambda_{J'}\}, x, 1) - \mathit{SE}(\lambda_J, \lambda_{J'})| < \varepsilon, \tag{2.5}$$

and

$$\max_{x \in [\lambda_J, \lambda_{J'}]} |\mathit{ASE}(\{\lambda_J, \lambda_{J'}\}, x, 1) - \mathit{SE}(\lambda_J, \lambda_{J'})| < 2\varepsilon. \tag{2.6}$$

IDEA OF THE PROOF: Figure 2.16(a) illustrates the worst case scenario of the approximate shape error. The estimation error is very skewed: in interval $[\lambda_J, x]$ it is higher than ε and in interval $[x, \lambda_{J'}]$ it is lower than ε. The average error over both intervals can be in the worst case ε, while the worst case is bounded by 2ε. □



(a) Worst Case Scenario for the approximated shape error

(b) Estimation of Initial Partitioning from a Histogram

Figure 2.16: Approximated Shape Error and the Initial Partitioning (1D)

Investigation of the error of approximate shape error in higher dimensional case is analogous to the one-dimensional case, and a similar result can be formalized:

Theorem 2.4. *Let C be a cell such that the density is monotonically increasing or decreasing in C, and the second derivative does not change the sign in C ($f''(\mathbf{x}) < 0$ or $f''(\mathbf{x}) > 0$). Then, on average average shape error approximates shape error:*

$$\frac{1}{|\mathsf{C}|} \int_{\mathbf{x} \in \mathsf{C}} |\mathit{ASE}(\mathsf{C}, x, i) - \mathit{SE}(\lambda_J, \lambda_{J'})| < \varepsilon, \tag{2.7}$$

and

$$\max_{\mathbf{x} \in \mathsf{C}} |\mathit{ASE}(\mathsf{C}, x, i) - \mathit{SE}(\lambda_J, \lambda_{J'})| < 2\varepsilon, \tag{2.8}$$

where $\lambda_J, \lambda_{J'} \in \mathsf{C}$ are two adjacent grid points.

Databases usually create indexes (histograms or B-trees) for most commonly queried attributes. These indexes can be used to approximate the initial partition.

Figure 2.16(b) illustrates the approximation of the initial partitioning. If an equi-width histogram is given then one should allocate partition points at the local minima and local maxima points of the density. If equi-height histogram is given one can compute the density of the equi-width histogram in the following way. One should compute the width w for each bin b; the equi-width value at bin b is then $1/w$. Note, that a b-tree can also be used to approximate the initial partitioning, because each level of the tree correspond to an equi-height histogram.

2.7 Approximate Answering of Aggregate Range Queries

Queries of the form "total number of orders of customers which age is between 20-30", "monthly income between 1500-5000Eur", "weekly spending not less than 50Eur on gym" or "maximum number of working hours of any doctor in last 20 days" are examples of aggregate queries. Given a d-dimensional query frame aggregate queries aggregate the data points within the frame (compute the COUNT, SUM, AVG, MIN, or MAX).

Computation of precise aggregate queries can be computationally expensive for large databases, since all data points in the query frame has to be examined. In applications where exact answer of the aggregate query is not required approximate aggregates can be computed trading a bit of accuracy for a fast computation of the answer. Typical examples of such applications are selectivity estimation, OLAP, data warehousing, exploratory analysis, and data mining.

This section shows how to approximate aggregate queries using the AD-Tree. There are two key challenges to approximate aggregate queries from the AD-Tree. First, differently from other data summary structures, the AD-Tree stores continuous density information and, hence, the aggregate operators are defined by a multiple integration of density function. Second, the AD-Tree do not aggregate the data of their grids at different levels of the hierarchy, but increases the precision of the density. Any grid that fall into the query range may participate in answering the query, however, only the unsplit cells of the grids provide the most precise answer. The AD-Tree does not store the splitting information of grids for compact representation of the tree and, therefore, this information must be derived from the hierarchy of the grids.

This Section makes the following contributions:

- We prove that answering an aggregate query from the AD-Tree can be rewritten as a multiple integral. The multiple integral is efficiently computed with by linearly interpolating the density values on a limited number

of points. Specifically, the number of points correspond to the number of cell in the AD-Tree which fall into a query range.

- We develop two algorithms: 'Range Split' and 'Bitmap' to answer aggregate queries approximately. The algorithms have different best and worst case scenarios. The 'Range Split' algorithm benefits when a large number of cells fall into the query range and the 'Bitmap' algorithm is preferable for cases when grids in the query range has many child grids. Depending on the query parameters we automatically select the best algorithm for query answering.
- We experimentally compare querying the AD-Tree. Our results show that the AD-Tree is more accurate and at least as fast as GenHist and wavelets.

The rest of this section is organized as following. In Section 2.7.1 we in details discuss challenges of querying the AD-Tree. Sections 2.7.2, 2.7.3 present our solution on finding unsplit cells, computation of their intersection with a query range and computation of the multiple integral. We present the experiments on querying in Section 2.9.6 together with all experiments on the AD-Tree.

2.7.1 Aggregate Queries as Functions of the Density

The AD-Tree stores density f of the data at the grid points. We write aggregate operators COUNT, SUM, and AVG as multiple integration of f. Let d-dimensional frame F_q be a range of aggregate query and n be the number of tuples in the dataset. Then the number of tuples in query range F_q is:

$$COUNT(\mathsf{F}_q) = n \cdot \int_{\mathsf{F}_q} \cdots \int f([x]_1, \cdots, [x]_d)\, \mathrm{d}[x]_1 \ldots \mathrm{d}[x]_d,$$

the sum and average of the k-th attribute values in query range Q is:

$$SUM(\mathsf{F}_q, k) = n \cdot \int_{\mathsf{F}_q} \cdots \int [x]_k \cdot f([x]_1, \cdots, [x]_d)\, \mathrm{d}[x]_1 \ldots \mathrm{d}[x]_d,$$

$$AVG(\mathsf{F}_q, k) = \frac{SUM(\mathsf{F}_q, k)}{COUNT(\mathsf{F}_q)}.$$

The following definition and lemma formalizes aggregate operators over AD-Tree:

Definition 2.9. *[Cell-Query Intersection]. Let $\mathsf{F}_q = (P, Q)$ be a d-dimensional query frame. Let G^k be a grid of d-dimensional AD-Tree. Then, intersection of cell $\mathsf{C}^k_J \subseteq \mathsf{G}^k$ with query range F_q is d-dimensional frame (P^k_J, Q^K_J) satisfying:*

$$[P^k_J]_i = \min(\max([\lambda^k_J]_i, [P]_i),$$
$$[Q^k_J]_i = \max(\min([\lambda^k_{J+\mathbb{I}(i)}]_i, [Q]_i), [\lambda^k_J]_i),$$

for all $1 \leq i \leq d$.

Example 2.4. *Let* $\mathsf{C}^1_{2,2} = \{(0.25, 0.25), (0.375, 0.25), (0.25, 0.375), (0.375, 0.375)\}$ *be a cell and* $(P, Q) = ((0.3, 0.3), (0.8, 0.8))$ *be the query frame. The cell-query intersection is*

$$[P^1_{2,2}]_0 = \min(\max(0.25, 0.3), 0.375) = 0.25$$
$$[Q^1_{2,2}]_0 = \max(\min(0.375, 0.8), 0.3) = 0.375$$
$$[P^1_{2,2}]_1 = \min(\max(0.25, 0.3), 0.375) = 0.25$$
$$[Q^1_{2,2}]_1 = \max(\min(0.375, 0.8), 0.3) = 0.375.$$

Lemma 2.7.1. Let I_g denote the multiple integral of any aggregate operator. Then, for any query range F_q, $I_g(\mathsf{F}_q)$ is the sum of $I_g(P^k_J, Q^k_J)$ for all nonempty cell-query intersections (P^k_J, Q^k_J) such that C^k_J are not split:

$$I_g(\mathsf{F}_q) = \sum_{\substack{\mathsf{C}^k_J \text{ not split,} \\ (P^k_J, Q^k_J) \text{ not empty}}} I_g(P^k_J, Q^k_j)$$

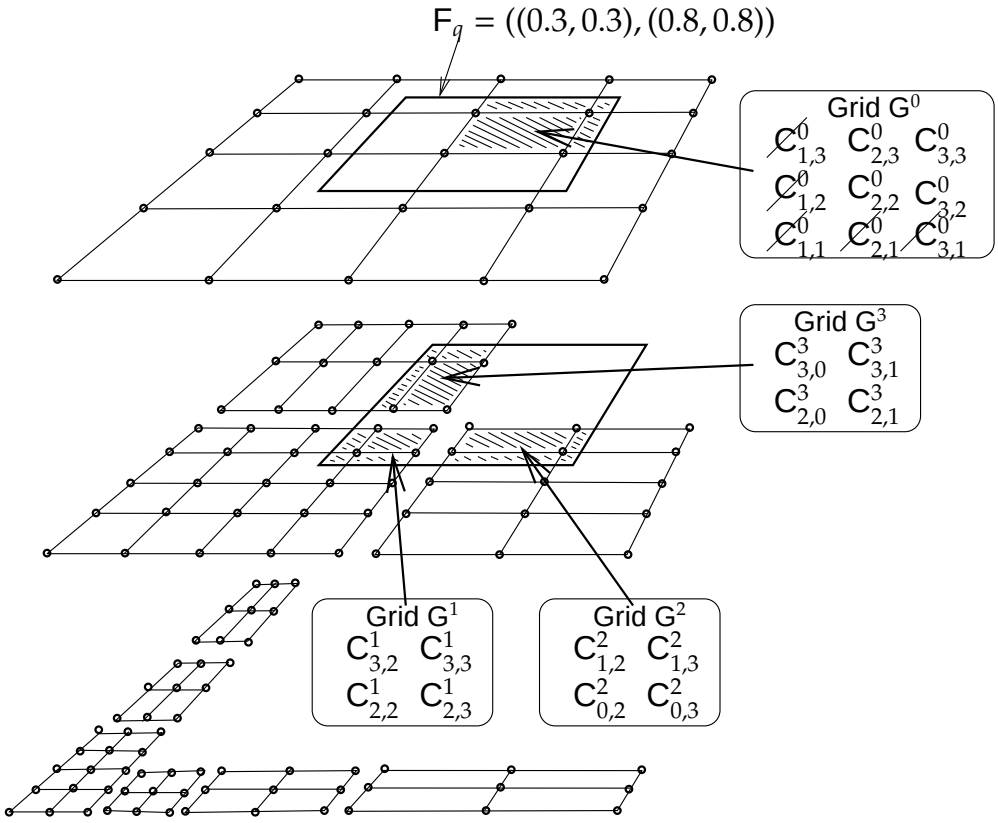


Figure 2.17: The Challenge of Finding Unsplit Cells which Intersect with the Query Range.

According to the lemma above, for an accurate computation of the aggregate queries we need to compute the multiple integrals only over the cells which are not split further. Otherwise, we would get an under- or an overestimation of the aggregate query in the overlap areas between grids at different levels of the hierarchy. In the AD-Tree the non-overlap areas of the grids correspond to non-split cells. Consider Figure 2.17 as an example. The input query range $\mathsf{F}_q = ((0.3, 0.3), (0.8, 0.8))$ intersects with grids G^0, G^1, G^2 and G^3. Grids G^1, G^2 and G^3 split cells of grid G^0 and, hence, are child grids of G^0. The overlap area of child grids G^1, G^2 and G^3 with grid G^0 within the query range corresponds to split cells of G^0, i.e.: cells $\mathsf{C}^0_{1,3}$, $\mathsf{C}^0_{1,2}$, $\mathsf{C}^0_{1,1}$, $\mathsf{C}^0_{2,1}$ and $\mathsf{C}^0_{3,1}$. For an accurate computation of the aggregate query we must not compute multiple integral over the split cells but, only over the unsplit cells of the child grids. Otherwise, if we compute multiple integral over all split and unsplit cells we get overestimated answer for the aggregate query. If we compute multiple integral over the cells of one grid G^0 we get underestimated answer for the aggregate range query.

Summarizing the above, there are two challenges in computing aggregate range queries with the AD-Tree. First, we need to find unsplit cells within a query range and for each compute cell-query intersection. Second, we need evaluate multiple integral of the density function over each cell-query intersection. Next, we develop efficient solutions for both challenges.

2.7.2 Finding Unsplit Cells and Computing Their Intersection Area with a Query Range.

The key challenge in finding unsplit cells and computing their intersection areas with the query range, is to minimize the number of computations spend on retrieving intersections between frames. For example, in order two check if a cell of a grid with m child grids is split, we need to traverse all m child grids and compute intersections of their frame with the frame of the child. In this section we develop two algorithms to find unsplit cell and their intersection area with a query range. The 'Range Split' algorithm iteratively splits the query range into smaller subranges until each subrange encloses only unsplit cells of the grid. The bitmap algorithm creates a d-dimensional bitmap. For each cell that fall into a query range a full bit indicates that the cell is split. The algorithms have different complexity. In the best case, the 'Range Split' algorithm computes intersection between two frames only ones. In the worst case, number of intersections computed by the 'Range Split' algorithm is quadratic wrt the number of child grids. Complexity of the 'Bitmap' algorithm is linear wrt the number of child grids, however, the 'Bitmap' algorithm is affected by the number of cells. The 'Range Split' algorithm is more preferable for cases when many cells fall into the query range but there are only a few child grids. The 'Bitmap' algorithm benefits in cases when the node has a big number of child grids. We adaptively select the

'Range Split' or 'Bitmap' algorithms depending on the characteristics of the grid.

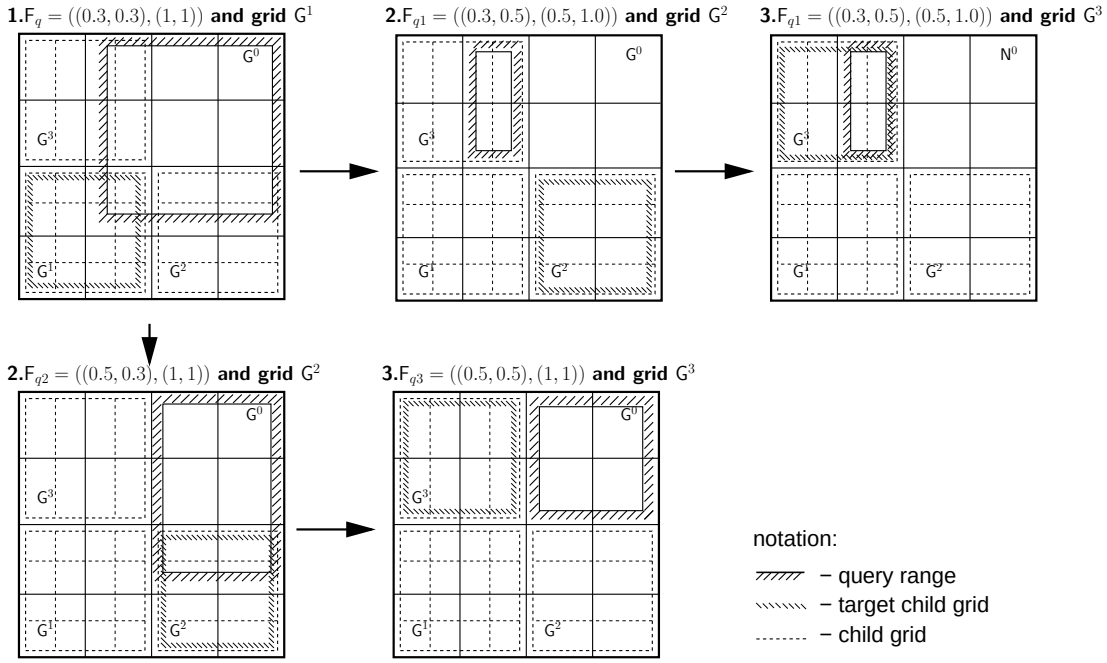


Figure 2.18: Iterations of the 'Range Split' Algorithm. Each Iteration Divides Query Range into SubRange so that they do not Intersect with the Target Child Grid.

Figure 2.18 illustrates the iterations of the 'Range Split' algorithm on a two dimensional grid G^0. Grid G^0 has three child grids G^1, G^2 and G^3 which split its cells in three corners except the top-right corner. Query range F_q encloses both split and unsplit cells. The 'Range Split' algorithm finds unsplit cells and computes their intersection area with the query range by iteratively splitting the query range into subranges. At each iteration the 'Range Split' algorithm computes the intersections of (sub)ranges with one child grid: it splits each (sub)range that intersect with a child grid into new subranges so that none of them overlaps with the frame of a child grid. For example, during the first iteration illustrated in the Figure 2.18, the 'Range Split' algorithm split query range F_q into new subranges F_{q1} and F_{q2} so that they do not intersect with the child grid G^0. In the next iteration the split algorithm proceeds with the next child grid and all subranges or quits if there are no subranges or child grids to proceed on. For example, during the second iteration illustrated in the Figure 2.18, the 'Range Split' algorithm proceeds with the next child grid G^2. Grid G^2 intersects only with subrange F_{q2} and, hence, the 'Range Split' algorithm split F_{q2} into one new subrange F_{q3}. Subrange F_{q1} encloses only cells of child grid G^3 and, hence, is eliminated during the third iterations. The algorithm terminates on fourth iteration with subrange F_{q3} which does not intersect with any child grid. The final output

of the 'Range Split' algorithm are query subranges, each enclosing only unsplit cells of a grid, and computed intersection area of each unsplit cell with the query range.

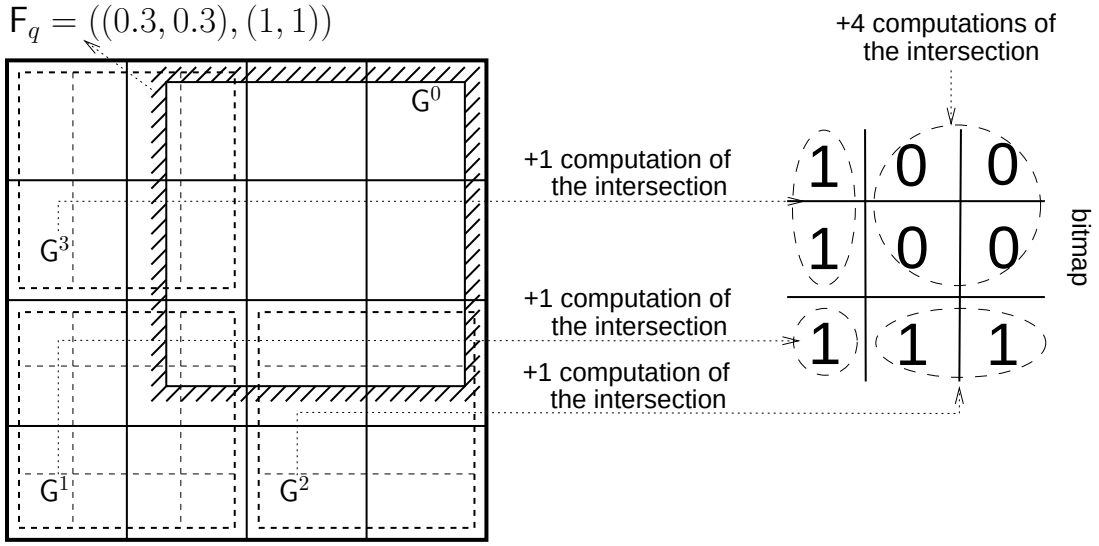


Figure 2.19: The 'Bitmap Split' Algorithm.

Figure 2.19 illustrates the 'Bitmap' algorithm for a two dimensional grid G^0 and query range $\mathsf{F}_q = ((0.3, 0.3), (1.0, 1.0))$ which encloses nine cells in the grid. The 'Bitmap' algorithm starts from the creation of a d-dimensional bitmap with number of bits equal to the number of cells in the query range. Initially, each bit is set to zero indicating that all cells in the query range are unsplit. Next, the 'Bitmap' algorithm computes unsplit cells and their intersection area with the query range in two steps. First, it iterates through all child grids. If a child grid intersects with the query range, then, by computing the intersection between the child grid and the query range, the 'Bitmap' algorithm identifies all cells which are split by the child grid and set the corresponding bits in the bitmap. Second, the 'Bitmap' algorithm iterates through cells in the query range and, if the corresponding bit in the bitmaps is zero, computes the intersection area between the cell and query range.

Theorem 2.5. *['Bitmap' Complexity]. Let* G *be the grid the of d-dimensional AD-Tree. Let* m *be the number of child grids produced by the dimensional split, unsplit and group steps of cells in* G*. Let* c *be the number of cells in grid* G *that fall into the query range* F_q*. Then, in the worst case the 'Bitmap' algorithm computes*

$$m + c$$

intersections of two frames.

PROOF: The worst case for the 'Bitmap' algorithm occurs when none of the m child grids intersects with the query range. Thus, the 'Bitmap' algorithm computes intersection of the query range with all m child grids and with all c cells. Figure 2.20(a) illustrates the worst case for the 'Bitmap' algorithm. □

Theorem 2.6. *['Range Split' Complexity]. Let* G *be the grid of the d-dimensional AD-Tree. Let m be the number of child grids produced by the dimensional split, unsplit and group steps of cells in* G*. Then, in the worst case the 'Range Split' computes*

$$m + \frac{m(m-1)(2d-1)}{2}$$

intersections of two frames.

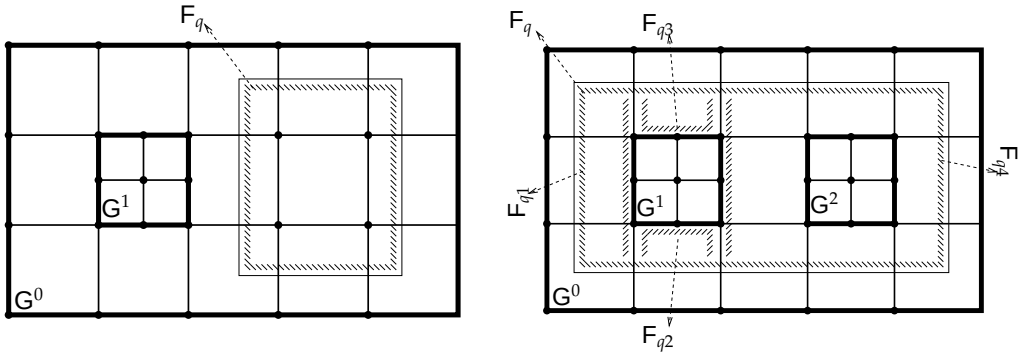
PROOF: The worst case for the 'Range Split' algorithm occurs when: *i*) all m child grids fall into the query range, *ii*) at each iteration the query range is split into the maximal number of subranges and *iii*) each iteration computes intersections with all subranges (no subranges are eliminated).

Since frame of a grid has $2d$ number of faces and 'Range Split' algorithm split the query range along the faces of grid's frame, hence, in the worst case $2d-1$ new query subranges are produced in one iteration. Next, if each iteration computes intersection with all previously produced subranges, then, the total number of computed intersections after i-th iteration is expressed with arithmetic progression:

$$a_i = a_{i-1} + (2d-1) \text{ and } a_1 = 1$$

, where a_i and a_{i-1} is the number of intersections computed after i-th and $(i-1)$-th iterations correspondingly, and $(2d-1)$ is the number of new query subranges. Computing the sum of the arithmetic progression for m elements proves the theorem. □

Besides, that in the worst case the 'Range Split' algorithm has quadratic complexity wrt to the number of child grids m, even in the worst case, it can perform better than the 'Bitmap' algorithm. Such situations are often present when the number of cells that fall into the query range is greater than the number of child grids. As an example let us consider Figure 2.20(b) which illustrates the worst case for 'Range Split' algorithm. The query range F_q encloses both child grids G^1 and G^2 of grid G^0. During the first iteration on grid G^0, the 'Range Split' algorithm computes intersection of G^1 with query range F_q and split it into four subranges F_{q1}, F_{q2}, F_{q3} and F_{q4}. During the second iteration the 'Range Split' algorithm computes intersection of G^2 with all four subranges. This happens because subrange F_{q4}, which actually intersects with G^2, is the last in the sequence of subranges. In total, the 'Range Split' algorithm 5 times computes intersection of two frames. For the same scenario in Figure 2.20(b), the 'Bitmap' algorithm 15 times computes intersection of two frames: 2 times when it computes intersections



(a) The Worst Case for the 'Bitmap' Algorithm

(b) The Worst Case for the 'Range Split' Algorithm

Figure 2.20: The Worst Case Scenario for the 'Bitmap' and 'Range Split' Algorithms.

of child grids with query range F_q and fills the bitmap, and 13 times when it computes intersection area between each unsplit cell and the query range.

For the best computation speed of the aggregate queries, we enforce the following strategy to select the appropriate algorithm. For each grid we compute the number of cells c that fall into a query range and compare it with a number of child grids m. If c is greater than $m(m-1)(2d-1)/2$, when we use the 'Range Split' algorithm. Otherwise, we use the 'Bitmap' algorithm.

Example 2.5. *[Choosing the Appropriate Algorithm]. Let us consider Figures 2.18 and 2.19 which illustrate the 'Range Split' and 'Bitmap' algorithms on a two-dimensional grid* G^0 *with three child grids* G^1, G^2 *and* G^3. *The query range encloses nine cells of grid* G^0. *Since, the number of cells c in the query range is not greater than* $m(m-1)(2d-1)/2$*:*

$$c = 9 \not> 9 = 3(3-1)(4-1)/2$$

we use the 'Range Split' algorithm and that is the correct choice. The 'Range Split' algorithm need 5 times and the 'Bitmap' algorithm 7 times to compute intersections between two frames.

2.7.3 Efficient Computation of Multiple Integrals

In this section we prove, that multiple integrals in the AD-Tree can be efficiently computed by a cheap linear interpolation without lost in the estimation precision.

The estimated density function in the AD-Tree is linear between grid points of a cell. Hence, the value of estimated density at arbitrary d-dimensional point x is computed with a help of the linear interpolation.

Definition 2.10. *[Point-query of the AD-Tree tree]. Let* C_j^k *be a cell. Let* $x \in \mathsf{F}$ *be a point in the frame of the cell. Density estimation at x from the AD-Tree*

using the linear interpolation is

$$\hat{f}(x) = \sum_{\lambda^k_{J'} \in \mathsf{C}^k_J} f(\lambda^k_{J'}) \times \prod_{i=1}^{d} \left(1 - \frac{|[x]_i - [\lambda^k_{J'}]_i|}{[\lambda^k_{J+\mathbb{I}(i)}]_i - [\lambda^k_J]_i}\right)$$

Theorem 2.7. *Let frame (P^k_J, Q^k_J) be a cell-query intersection. Then the multiple integral over (P^k_J, Q^k_J) of the estimated density is:*

$$\int_{P^k_J}^{Q^k_J} \cdots \int \hat{f}([x]_1, \cdots, [x]_d)\, \mathrm{d}[x]_1 \ldots \mathrm{d}[x]_d = v(P^k_J, Q^k_J) \cdot \hat{f}(m)$$

where $m = \left(\frac{[P^k_J]_1 + [Q^k_J]_1}{2}, \ldots, \frac{[P^k_J]_d + [Q^k_J]_d}{2}\right)$ *is the middle point of frame* (P^k_J, Q^k_J) *and* $v(P^k_J, Q^k_J) = \prod_{i=1}^{d}([Q^k_J]_i - [P^k_J]_i)$ *is the volume of frame* (P^k_J, Q^k_J).

PROOF: The proof of the theorem derives from the mean value theorem which states, that if function $g : [a, b] \to \mathbb{R}$ is an integrable positive function, then there exists a number y in (a, b) such that:

$$\int_a^b g(t)\mathrm{d}t = g(y)(b - a)$$

Next, we show that y is the middle point of frame (P^k_J, Q^k_J) while integrating $\hat{f}$ over (P^k_J, Q^k_J).

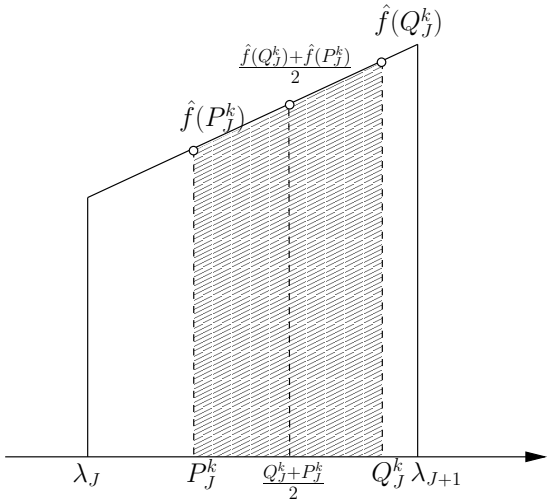


Figure 2.21: Illustration of Theorem 2.7 for One-Dimensional Case.

Let us consider Figure 2.21 which illustrates the theorem for one-dimensional case. The integral of $\hat{f}$ over frame (P^k_J, Q^k_J) is equal to the area of trapezoid

P_J^k_$\hat{f}(P_J^k)$_Q_J^k_$\hat{f}(Q_J^k)$:

$$\int_{P_j^k}^{Q_j^k} \hat{f}(x)\mathrm{d}x = \frac{\hat{f}(Q_J^k) + \hat{f}(P_J^k)}{2}(Q_j^k - P_J^k)$$

The observation, that $(Q_j^k - P_J^k)$ is the length of integration interval $[P_J^k, Q_J^k]$, and $(\hat{f}(Q_J^k) + \hat{f}(P_J^k))/2$ is the value of $\hat{f}$ in the middle point of the integration interval, i.e.:

$$\hat{f}\Big(\frac{Q_j^k + P_J^k}{2}\Big) = \frac{\hat{f}(P_J^k) + \hat{f}(Q_J^k)}{2},$$

proves the theorem for one-dimensional case. Investigation of the theorem in n-dimensional case is reduced two one-dimensional case since (P_J^k, Q_J^k) is a frame with faces perpendicular to axes and, hence, multiple integral of $\hat{f}$ can be decomposed into a product of one-variable integrals. □

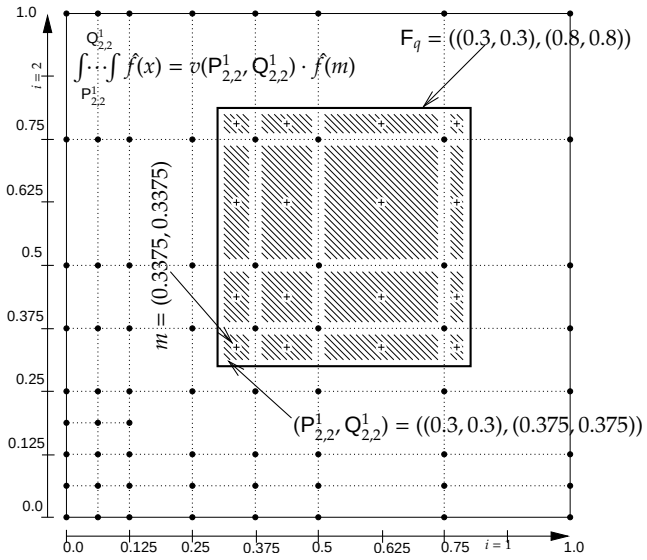


Figure 2.22: Efficient Computation of the Multiple Integral.

Figure 2.22 applies Theorem 2.7 for query range $\mathsf{F}_q = ((0.3, 0.3), (0.8, 0.8))$. Query range F_q encloses 16 unsplit cells of the AD-Tree. Multiple integral of $\hat{f}$ over query range F_q is a sum of 16 multiple integrals computed over each cell-query intersection of the unsplit cells. We efficiently compute multiple integral over cell-query intersection by multiplying its volume with the value of the estimated density function computed at the middle point of the cell-query intersection.

2.8 Algorithms and Complexity

AD-Tree method consists of five algorithms: *constructADTree*, *dimsplit*, *unsplit*, *merge* and *kernels* We present the algorithms in turn.

ConstructADTree (cf. Algorithm 1) constructs the AD-Tree from random d-dimensional sample D' of the dataset. The algorithm inputs two parameters: the estimation precision ε and initial partitioning G^I Ideally, the initial grid should be chosen based on the extrema points of the density (cf. Section 2.6). In practice, an initial grid of granularity 2 or 3 in each dimension i can be taken.

During the initial step (cf. Line 1 in Algorithm 1) the algorithm performs one scan of the input data and computes kernel additions on the initial grid. The initial grid becomes the root node of the AD-Tree. Each subsequent iteration (Section 2.4) is a sequence of five steps: dimensional split, group (Lines 5–7 in the Algorithm), kernel additions (Line 8), unsplit and group (Lines 9–12). The dimensional split introduces new grids only in the cells and only in the dimensions of high shape error. This ensures a minimal intermediate memory. The group step organizes the grid points into memory efficient large grids. The dimensional split of individual cells is not materialized. In contrast, the dimensional split and the group steps are combined in the algorithm. First all cells are marked with split dimensions and then grouped cells are introduced for all split cells. This reduces stress of the memory management component.

2.8.1 Dimensional Split and Unsplit

Dimsplit and unsplit are given in Algorithms 2 and 3. Both dimensional split and unsplit are integrated together with the group step for faster and more memory-efficient implementation. In both algorithms we maintain a d-dimensional bit array for each cell of the grid (cf. $[B_J]$ on Line 4 in Algorithm 2 and the same line in Algorithm 3). The algorithms gather the split information for each cell and at the end of the of the execution pass the split information into our group algorithm. The group algorithm computes and materializes larger grids.

Algorithms 2 and 3 avoid multiple computation of the approximated shape error on the same grid point. According to Definitions 2.3 and 2.6, at each grid point of a cell and in each dimension i we compute the approximated shape error. However, adjacent cells share grid points which yields to a multiple computation of the approximated shape error. In a d-dimensional grid a grid point may be shared by 2^d number of cells and that results on a multiple computation of 2^d approximated shape errors. Algorithms 2 and 3 have linear time complexity: they perform one scan of a grid and at each grid point they compute the approximated shape error only once.

2.8.2 Group Step

The algorithm for the group step is presented in Algorithm 4. The algorithm inputs grid G and bit array B for the all cells in the grid. The complexity of the algorithm is linear wrt to the number of cells in grid G.

Algorithm 1: constructADTree($D' \subseteq D$, ε, G^I)

```
   /* compute kernel additions on initial grid                          */
 1 for X^l ∈ D' do kernels(G^I, X^l);

   /* compute the AD-Tree                                               */
 2 CurrentLevel = {G^I};
 3 while CurrentLevel ≠ ∅ do
 4     NewLevel = ∅;

       /* split grids and group the output into maximal grids           */
 5     for G^k ∈ CurrentLevel do
 6         NewLevel = NewLevel∪dimsplit(G^k, ε);
 7     end

       /* compute kernel additions on new grids                         */
 8     for X^l ∈ D' do kernels(NewLevel, X^l);

       /* remove unnecessary splits and group the output into
          maximal grids                                                 */
 9     for G^k ∈ CurrentLevel do
10         M = {Ĝ^m ∈ NewLevel : Ĉ^m_ĵ = ds(C^k_J ∈ G^k) for all Ĉ^m_ĵ ∈ Ĝ^m};
11         NewLevel = (NewLevel − M)∪unsplit(G^k, M, ε);
12     end

       /* store new level of the AD-Tree                                */
13     CurrentLevel = NewLevel;
14 end
```

Algorithm 4 iteratively computes frames of larger grids. At each iteration the algorithm takes the smallest available index J_{min} (cf. Line 3) and tries to expand in all directions as far as possible (Lines 5–9). This ensures that frame F_{max} is of maximal grid and encloses only the cells of the input grid G of the same split. Eventually we materialize the larger grid (Lines 10–14) and remove the cells from the processing (Line 15).

2.8.3 Kernel Additions

The direct implementation of kernel addition (Equation 2.1) is for each grid point scan the sample and augment the kernels on the grid point. This is computationally expensive and requires to scan over the sample for each grid point. The organization of the grid points into grids allows to optimize computation of kernel additions and decrease the number of scans. The optimization is based on the fact that the kernel function is non zero in a bounded space. Therefore, a data point

Algorithm 2: dimsplit(G, ε)

```
   /* iterate through the grid points of the input grid          */
1  for λ ∈ G do
2      for i : 1 ≤ i ≤ d do
           /* if the approximated shape error at λ is greater than
              ε set the ith bits of all cells containing λ        */
3          if ASE(G, λ, i) > ε then
4              for C_J ∈ G : λ ∈ C_J do [B_J]_i = 1;
5          end
6      end
7  end
8  return group(G, B);
```

may add up kernel values only to a limited number of grid points as illustrated in Figure 2.23. The two dimensional parabola corresponds to the Epanechnikov's kernel at data point X^l. The radius of the base of the parabola is h_{opt} (cf. Section 2.3). Grid points which are covered by the parabola are influenced by data point X^l, i.e. kernel placed at X^l is positive at these grid points and, hence, contributes to the final density value at these grid points. All other grid points are not influenced by data point X^l and the contribution of the kernel at X^l to the density at these grid points is zero.

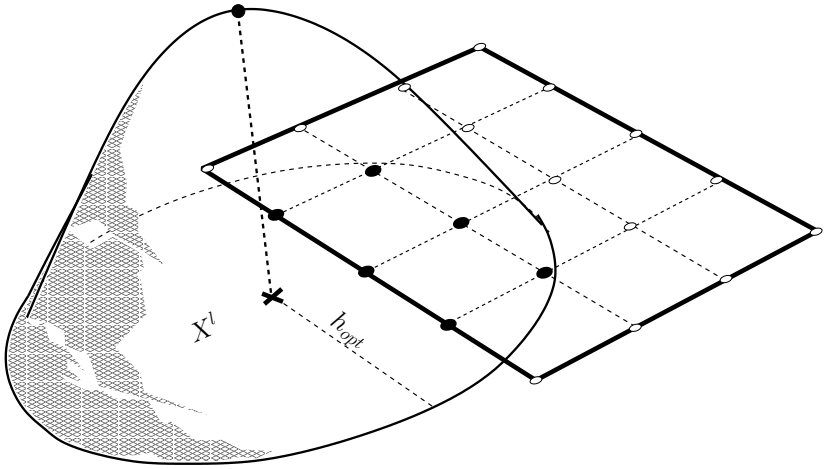


Figure 2.23: Optimization of Kernel Additions.

Algorithm 5 summarizes the optimized computation of the kernel additions. The algorithm inputs a set of grids and data point X^l and computes the influence range J_{min} and J_{max} by data point X^l. Finally, the algorithm iterates through the influenced grid points and augments the kernels with $K((\lambda_J^k - X^l)/h_{opt})$.

Algorithm 3: unsplit(G^p, $\{\mathsf{G}^{c_0}, \mathsf{G}^{c_1}, \ldots, \mathsf{G}^{c_k}\}$, ε)

```
  /* iterate through child grids                                          */
1 for {G^{c_l}} do
    /* iterate through the grid points of G^{c_l} excluding the grid
       points co-occurring in G^p                                         */
2   for λ ∈ G^{c_l} \ G^p do
3     for i : 1 ≤ i ≤ d do
        /* if the approximated shape error and is greater
           than ε set the appropriate bit                                 */
4       if ASE(G^{c_l}, λ, i) > ε then [B_J]_i = 1 for all J : λ ∈ C_J ⊂ G^{c_l};
5     end
6   end
7 end

8 return group ({G^{c_1}, ..., G^{c_k}}, B);
```

2.9 Experiments

This Section evaluates the AD-Tree experimentally. We show the minimal intermediate memory usage and efficient organization of the high dimensional points in three ways. In Section 2.9.1 we show that the decrease of the shape error as the number of iterations increases. The AD-Tree does not split the cells of the tree if the shape error in the cell is already below the threshold. This reduces the requirements for the intermediate memory substantially. In Section 2.9.2 we investigate the effectiveness of the dimensional split. The comparison of the dimensional split versus the full split of the cells that require a split decreases the memory requirements by at least of an order of magnitude and the effectiveness increases as the dimensionality of the data increases. Comparison between the AD-Tree and non-minimal intermediate memory techniques is given in Section 2.9.3. We evaluate the approximation of the shape error in Section 2.9.4 and show the effectiveness of the AD-Tree on the real world data in Section 2.9.5. Finally, Section 2.9.6 presents experiments evaluating the AD-Tree for computing aggregate range queries.

Most of the experiments are produced on the three-dimensional spiral dataset (cf. Figure 2.24). We used one-dimensional spiral dataset to illustrate the decrease of the shape error as the number of iteration increases (cf. Figure 2.25, Section 2.9.1). In Section 2.9.2 evaluates the method for a two-dimensional and three-dimensional datasets consisting of a one-dimensional line.

Algorithm 4: group(G , B)

```
  /* initialize the set of larger grids                              */
1 M = ∅;

  /* proceed until there are unprocessed cells                       */
2 while B_J^k ∈ B : B_J^k ≠ 0 do
    /* start with the smallest available index                       */
3   J_min = J_max = min{J : B_J^k ≠ 0};
    /* expand in all dimensions at the beginning                     */
4   e = ([e]_1, ..., [e]_d) = (true, ..., true);

    /* iterate until there are no dimensions to expand               */
5   while ∃i : [e]_i = true do
6     if B_J^k = B_{J_min}^k for each J_min ≤ J ≤ J_max + 𝕀(i) : B_J^k ∈ B then
7       J_max = J_max + 𝕀(i);
8     else e_i = false;
9   end

    /* create and add new larger grid to the output set              */
10  J_max = J_max + 𝕀;
11  F_max = F(λ_{J_min}, λ_{J_max});
12  S_max =
    (([B_{J_min}]_1 + 1) · [J_max − J_min]_1, ..., ([B_{J_min}]_d + 1) · [J_max − J_min]_d);
13  M = M ∪ G(F_max, S_max);

14  B_J^k = 0 for all J_min ≤ J < J_max : B_J^k ∈ B
15 end

16 return M;
```

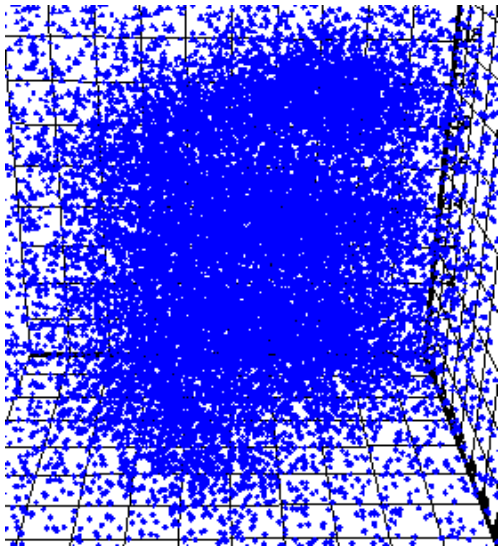


Figure 2.24: Spiral Dataset.

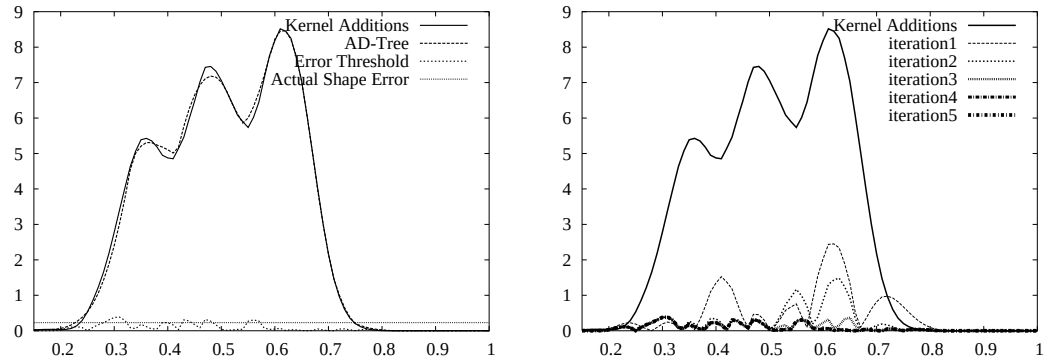
Algorithm 5: kernels($\{\mathsf{G}^0, \mathsf{G}^1, \ldots, \mathsf{G}^k\}$, X^l)

```
1 for G^k do
      /* compute the influence range of X^l                    */
2     J_min = min{J : |λ_J^k − X^l| < h_opt};
3     J_max = max{J : |λ_J^k − X^l| < h_opt};

      /* augment kernels                                       */
4     for J : J_min ≤ J ≤ J_max do  f(λ_J^l) = f(λ_J^l) + K((λ_J^l − X^l)/h_opt);
5 end
```

2.9.1 Uniform Control of the Shape Error

Shape error enables the minimal intermediate memory of the AD-Tree. The AD-Tree method identifies the cells where the shape error is higher than the given threshold and adds grids to more accurately describe the density of the data. The actual shape error for the spiral dataset (cf. Figure 2.25(a)) is around the given threshold showing the effectiveness of the dimensional split. Since the shape error is approximated, the actual shape error slightly exceeds the required threshold in certain areas of the domain.



(a) Shape Error of the AD-Tree

(b) Decrease of the Shape Error After Each Iteration

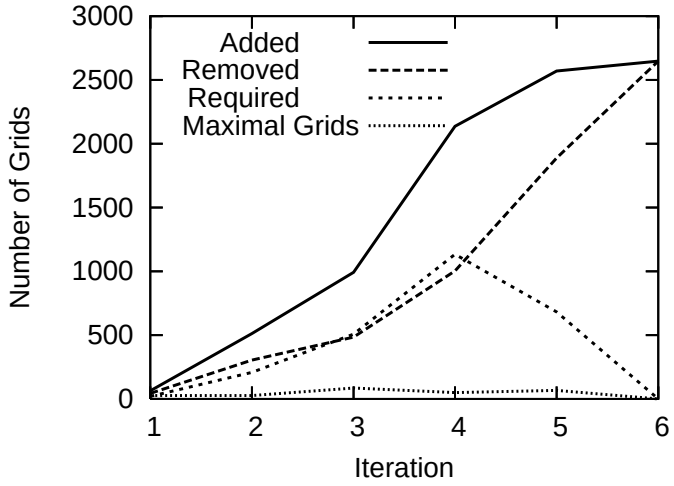
Figure 2.25: Shape Error for the Spiral Dataset and threshold $\varepsilon = 0.1$

Figure 2.25(b) shows the decrease of the shape error after each iteration for the spiral dataset. First iteration puts new grid points in the whole domain. After that, the shape error is low at the left border of the domain and is aggressively high in two intervals $[0.37, 0.44]$ and $[0.5, 0.8]$. Grids introduced by the second and third iteration more than twice decreases the shape error in these regions. The iterative process stops when the approximated shape error is lower than the required threshold resulting in the uniformly good approximation of the function.

Figure 2.26 evaluates the construction of the AD-Tree numerically. Organization of the grid points into a tree data structure results in a small overhead in the construction of the summary structure. Most computational time is spent for kernel additions and only very little is spent on computing and organizing new grid points in a tree like data structure (cf. the first line in the table of Figure 2.26(a)).

	1	2	3	4	5	6
Kernel Additions	0.31	0.50	1.34	3.00	6.75	4.76
Dim.Split, Unsplit, Group	0.00	0.00	0.01	0.01	0.01	0.02
Overall	0.31	0.50	1.35	3.01	6.76	4.78

(a) Time



(b) Evaluation of the Tree Maintenance Operations

Figure 2.26: Memory and Time spent at the End of Each Iteration

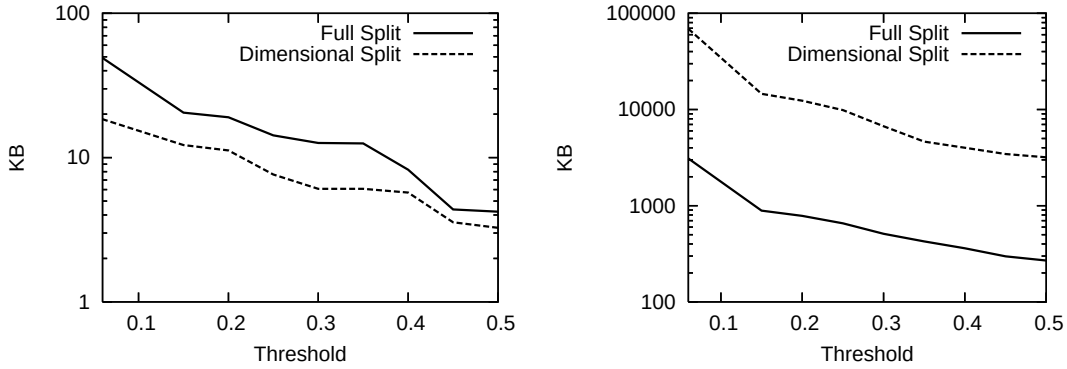
Organization of the individual grids into larger grids is an essential and very effective step of the AD-Tree especially as the number of the iterations increases (cf. Figure 2.26(b)). For example, more than 2000 grids are introduced by the dimensional split in iteration 4, around 40% of the grids are removed due to the overestimation of the approximate shape error, and the remaining required small grids are grouped only into 20–80 larger grids.

2.9.2 Dimensional Split

In this Section we evaluate the effectiveness of the dimensional split. We generated a two-dimensional (cf. Figure 2.27(a)) and a three-dimensional dataset (cf. Figure 2.27(b)) containing one dimensional line.

Figure 2.27 evaluates the AD-Tree for datasets consisting of low dimensional structures. The difference (cf. the dashed line for the dimensional split and solid line for a non-dimensional split in the Figures) shows the increase of the

effectiveness of the dimensional split as the threshold ε and the dimensionality of the dataset increases.



(a) Two-Dimensional Data (b) Three-Dimensional Data

Figure 2.27: Memory Usage of the AD-Tree for Two- and Three-Dimensional Datasets with a One-Dimensional Structure

We draw two conclusions from the experiment. First, the memory complexity of the AD-Tree increases exponentially as the number of dimensions d increases (cf. the scale of the Y Axis, Figure 2.27). The AD-Tree partitions the d-dimensional space into a hierarchy of d dimensional grids and computes d-dimensional kernel additions on the partition points. Since all the points are d-dimensional, the increase is exponential as the number of dimensions increases. Second, the AD-Tree adapts to the dimensionality of the structures. For two-dimensional dataset, the dimensional split reduces the memory usage by 2–5 times. For three-dimensional dataset, the dimensional split reduces the memory usage by 20–50 times. The higher is the difference between the dimensionality of the dataset and the dimensionality of the structures, the bigger is the reduction of the memory. Our experimental results show that the decrease of memory by the dimensional split is exponential wrt d.

2.9.3 Related Techniques of High Intermediate Memory

Because no other kernel based density data summary structures are available for direct comparison we compare the AD-Tree with a generalization of the most common type of kernel estimator implemented on a uniform partitioning (UP) with a fixed number of partition points. The conceptual simplicity makes UP an attractive candidate. Note, that conclusions of the section are generalized to equi-width histograms and wavelets: equi-width histograms allocate buckets uniformly at cells of UP and wavelets compress equi-width histograms. We use the three-dimensional spiral data set in our experiments.

We use two interpolation functions with the UP data structure: the constant interpolation (UPC method) and the coordinate-wise linear function (UPL

method). UPC assumes a stepwise constant function to interpolate the density values between partition points. UPL uses the coordinate-wise linear function to interpolate the density in between the grid points. Both UPC nor UPL allocates a uniform grid and computes the kernel additions on it. We allocate such a grid that the estimation error is below the given threshold in the whole domain. Note, that neither UPC nor UPL cannot compute the size of the grid that the estimation error is below the given threshold. In the experiment we had to derive this information from the AD-Tree. For the UPL we choose the number of grid points so the distance between any two consequent partition points is equal to the smallest distance between any two consequent grid points in the AD-Tree:

$$N^{\mathrm{UPL}} = \Big(\min_{\lambda_J, \lambda_{J'}} \big\{ |\lambda_J - \lambda_{J'}| \big\} \Big)^{-d},$$

where the minimum is computed over all leaf grid points. The number of grid points for the UPC method is calculated so the difference of the values of the density between any two consequent grid points is less than the given threshold:

$$N^{\mathrm{UPC}} = \Big(\min_{\lambda_J, \lambda_{J'}} \big\{ |(\lambda_J - \lambda_{J'})/(f(\lambda_J) - f(\lambda_{J'}))|/\varepsilon + 1 \big\} \Big)^{-d}.$$

Experimental memory complexity for different estimation precisions is given in Table 2.2. Since the UP data structures allocate the partition points uniformly, the estimation results in a very skewed distribution of the shape error. In some cells the shape error is very small, while in other cells it is just below the threshold. Overall, the uniform condition of the UP methods requires to allocate a very fine grid, resulting in dramatic increase of memory usage, as the error decreases. The UPC method is impractical even for a relatively large threshold. The UPL method performs a lot better, however is almost two orders magnitude worse than AD-Tree.

ε	0.5	0.25	0.05	0.01
AD-Tree	0.02	29	233	2,607
UPL	0.04	256	8,256	65,792
UPC	0.33	129,313	$3.01 \cdot 10^8$	$1.47 \cdot 10^{11}$

Table 2.2: Memory Consumption in KB for Different Precisions

Table 2.3 shows the experimental time complexity for the UPL and the AD-Tree. We omit the timings for the UPC method, because it is infeasible to compute them. The AD-Tree computes the estimation of the density almost 2–3 times faster than the UPL method.

Table 2.4 shows the density computation time for a given query point x. We selected 10^6 random queries and averaged the results. The time complexity of the

ε	0.5	0.25	0.05	0.01
AD-Tree	0.31	5.06	63.81	497.40
UPL	0.33	8.92	289.98	1,574.47

Table 2.3: Computational Time in Seconds for Different Precisions

UPL method is constant, since one needs a constant time to identify the cell in the array the query point falls into, and a constant time to to apply the coordinate wise linear interpolation through the vertices of the cell to answer the query. The time complexity of the computation of the density value is logarithmic wrt the height of the tree in AD-Tree and therefore is higher for the AD-Tree compared to the UPL method. However, the computation overhead is not that big, and is only 2–3 times longer compared to the UPL method.

ε	0.5	0.25	0.05	0.01
AD-Tree	0.15	0.32	0.35	0.47
UPL	0.15	0.15	0.15	0.15

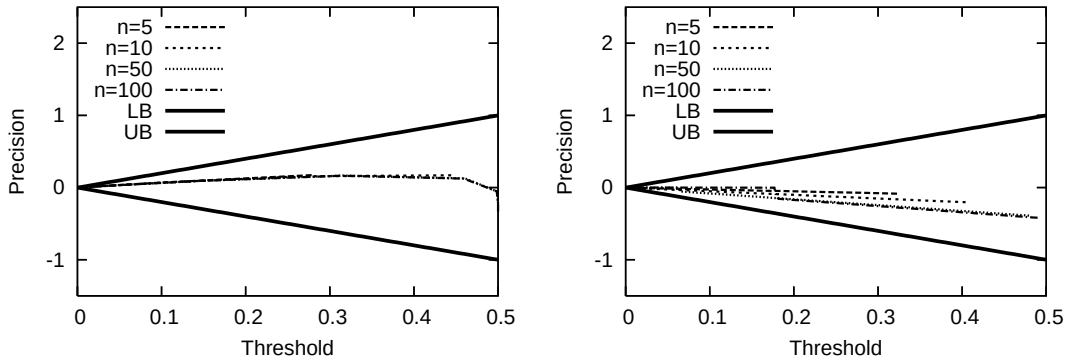
Table 2.4: Computational Time in Seconds for 10^6 Density Queries

2.9.4 Approximated Shape Error

This section evaluates the quality of the approximate shape error where the initial grid is set at the extrema points of the density. In the experiment we generated the density of the form $C_1(n) = (x_1x_2x_3)^n$, and $C_2(n) = (x_1x_2x_3)^{-n}$ in the unit cube $(x_1, x_2, x_3) \in [(0.0.0), (1, 1, 1)]$. We choose these two classes in the experiment, since they represent a general case of a density function that is monotonically increasing between the partition points (cf. Section 2.6.3).

Figure 2.28 evaluates the quality (the difference between the approximated and the true shape error). The solid line in the experiments indicate the theoretical bound 2ε of the estimator. According to Theorem 2.4 the approximated shape error varies in between these bounds.

We draw three conclusions from this experiment. First, that the actual approximation of the shape error is substantially better than the theoretical bound of 2ε. Second, there is almost a linear relationship between the approximated and true shape errors, i.e., $\mathsf{ASE} = c\mathsf{SE}$, where c is the slope constant of the linear relationship. Third, constant $c > 0$ for $C_1(n)$ ($\mathsf{ASE} < \mathsf{SE}$), and $c < 0$ ($\mathsf{ASE} > \mathsf{SE}$) for $C_2(n)$. Therefore, density queries of the AD-Tree are underestimated for $C_1(n)$, and overestimated for the class of $C_2(n)$.



(a) $C_1(n) = (x_1x_2x_3)^n$ (b) $C_2(n) = (x_1x_2x_3)^{-n}$

Figure 2.28: Approximated Shape Error

2.9.5 Real World Data

In this section we experimentally evaluate approximation quality, space and time complexity of AD-Tree on real world data Forest dataset. The dataset established itself as a comparison dataset for selectivity and density queries. The dataset consists of more than a half a million tuples and more than fifty numeric attributes. To asses the dimensional splits we use only three attributes of the dataset in combination with uniformly distributed additional attributes. The organization allows us to predict the density peaks and ensure that data distribution is the same as dimensionality changes.

We compare AD-Tree GenHist (the state of the art in histograms) and Haar wavelets (a good representative of wavelets) data summarization techniques. The results show that (i) for the same approximation precision the AD-Tree is faster to compute than Haar wavelets transformations and GenHist and (ii) the AD-Tree does not suffer from high intermediate memory complexity.

For each data projection in the Forest dataset we fix the size of data summary produced by Haar wavelets transformations and GenHist to the size of the AD-Tree that guarantees relative SE ≤ 0.01. Figure 2.29 illustrates the relation between the dimensionality of the data and the size of the AD-Tree.

Figure 2.30(a) compares the construction time of the AD-Tree, Haar wavelets and GenHist. The AD-Tree is faster to compute than GenHist and Haar wavelets transformations. The construction of the Haar wavelets is faster only for the three dimensional database. We could nod compute the Haar wavelets for databases of dimensionality higher than four, since it requires more than $4Gb$ of intermediate memory. Also, AD-Tree takes almost the same time to compute as GenHist.

Figure 2.30(b) compares the intermediate memory. Differently from the Haar wavelets the intermediate memory of AD-Tree is a lot lower. Comparing AD-Tree with Haar Wavelets transformations, for four dimensional data Haar Wavelets requires about $400Mb$ of intermediate memory. In contrast to that, for the same

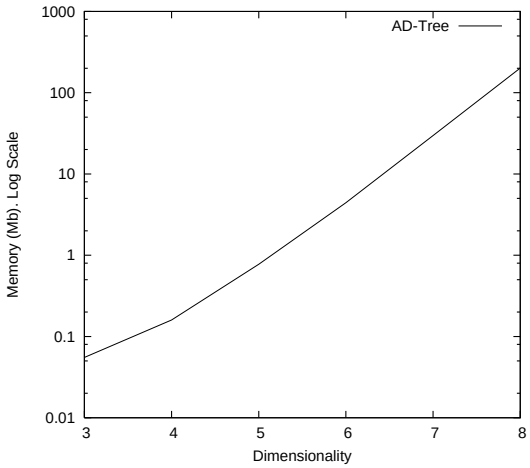


Figure 2.29: The Size of AD-Tree that Guarantees Relative for $\varepsilon = 0.01$

amount of $400Mb$ of intermediate memory, AD-Tree tree can be computed up to 8-dimensional data.

2.9.6 Approximate Aggregate Queries

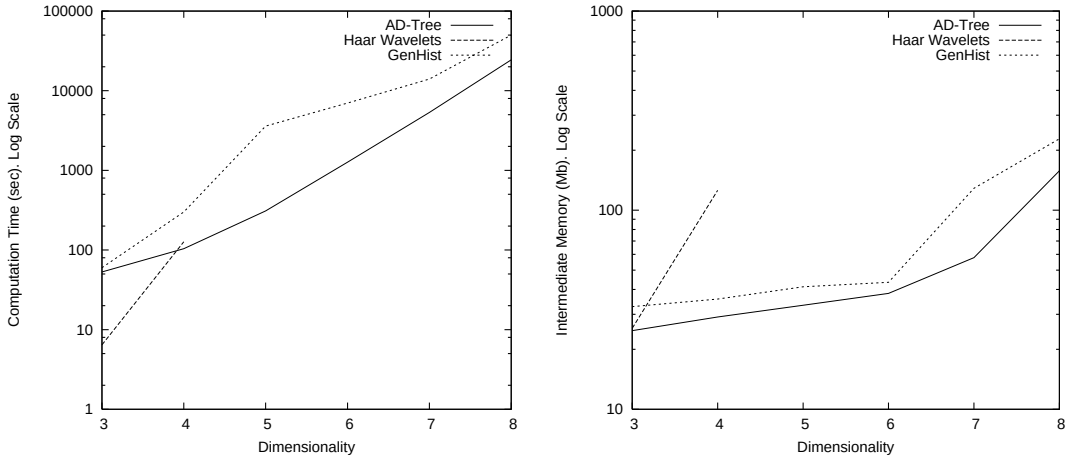
In this section we experimentally evaluate approximation quality and time complexity of the AD-Tree for multidimensional aggregate range queries. We compare the AD-Tree with GenHist and Haar wavelets approximate query answering techniques. Since both GenHist and Haar wavelets support approximation only of COUNT queries we limit the comparison only for the COUNT. We use the real world Forest dataset in the experiments.

The results show that (i) on a low-dimensional data the AD-Tree has query estimation accuracy close to the estimation accuracy of the Haar wavelets transformations, (ii) the AD-Tree is more accurate for high-dimensional data, (iii) the AD-Tree is faster compared to both Haar wavelets and GenHist.

We generated the query load in the following way: the dimensionality of the range is set to the dimensionality of the data, the true selectivity varies between 1% and 5%, the volume of the range is set from 1% to 70%, the number of multidimensional range queries is $10,000$. We reported the average relative error, i.e., let

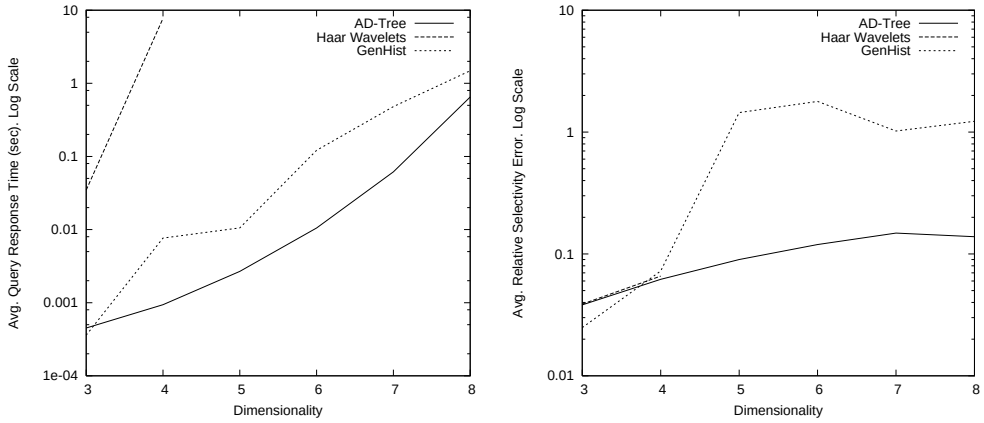
$P = \{p_0, p_1, ..., p_{1000}\}$ be a set of all queries in the workload, and S_i be the true and $\hat{S}_i$ be the estimated COUNT value of query p_i. Then average relative error is

$$\frac{1}{|P|} \sum_{i=0}^{|P|} \frac{|S_i - \hat{S}_i|}{S_i}$$



(a) Construction time (b) Intermediate memory

Figure 2.30: Comparison of Time and Memory Complexity



(a) Average query response time (b) Average relative error

Figure 2.31: Selectivity Estimation Computation Time and Quality

Figure 2.31(a) compares the average query time of the techniques. The AD-Tree is much faster than Haar wavelets. For dimensionality up to eight the AD-Tree tree response is less than a second. In contrast, already for the four dimensional case Haar wavelets require at least 10 seconds to estimate the query. GenHist shows faster results, however still slower that the AD-Tree. This is because GenHist need to consider all buckets within the query range even if the contributions of the buckets to the answer is small. The AD-Tree performs better because of the tree data structure: a lot fewer number of grid points are involved in to answer the query compared to GenHist.

Figure 2.31(b) compares the average relative error. The AD-Tree and Haar are very close to each other with almost the same accuracy. GenHist performs a

lot worse.

2.10 Summary and Future Work

The existing data summary structures suffer from high intermediate memory complexity. In this Chapter we develop the AD-Tree, a hierarchical summary structure for multidimensional data which ensures minimal consumptions of intermediate memory. The AD-Tree summarizes multidimensional data in terms of its density. It starts from a sparse initial grid that gives rough estimation of the density of the data. The AD-Tree iteratively increases precision of the estimation by placing new grid points along dimensions and areas where it increases the precision of the estimation. We measure the estimation quality with the help of the shape error. The shape error quantifies the deviation of the estimated density function from being linear on a one-dimensional grid. The dimensional split extends shape error for multi-dimensional grids: it iterates through cells of a grid and split them along dimensions of non-linearity of the estimated density function. The key challenges in answering aggregate range queries with the AD-Tree is computation of multidimensional integrals and finding unsplit cells. We prove that multidimensional integrals in the AD-Tree can be efficiently computed by a cheap linear interpolation without lost of the estimation precision. We develop two algorithms for finding unsplit cells and use them interchangeably depending on the parameters of the AD-Tree and query range. We prove optimality of the AD-Tree with respect to the initial partition, and we show that the AD-Tree adjusts to the local dimensionality of structures in the data and robustly approximates the shape error if the initial partition is chosen based on extrema points of the density. The experiments on real world and synthetics datasets confirm the analytical results.

There exists several future research directions. We plan to extend the AD-Tree with the update operation and apply our method on data streams and for indexing multidimensional data. At the technical level it would be interesting to eliminate the storage of the density measures at the same grid points in different levels of the hierarchy.

Chapter 3

CORE Clustering

Current clustering techniques are able to identify arbitrarily shaped clusters in the presence of noise, but depend on carefully chosen model parameters. The choice of model parameters is difficult: it depends on the data and the clustering technique at hand, and finding good model parameters often requires time consuming human interaction. In this chapter we propose CORE, a new nonparametric clustering technique that explicitly computes the local maxima of the density and represents them with cores. CORE proposes an adaptive grid and gradients to define and compute the cores of clusters. The incrementally constructed adaptive grid and the gradients make the identification of cores robust, scalable, and independent of small density fluctuations. Our experimental studies show that CORE without any carefully chosen model parameters produces better quality clustering than related techniques and is efficient for large datasets.

3.1 Introduction

CORE is a novel nonparametric clustering technique that explicitly computes and represents local density maxima. Neither the shape nor the dimensionality of these maxima is restricted. Modeling clusters in terms of explicitly computed local maxima permits clusters with two- or higher-dimensional maxima that are close to each other or overlap. Related clustering techniques do not explicitly represent local maxima. Instead they model clusters by, e.g., sets of unconnected one-dimensional maxima points [33], dense areas [17, 3], and statistical properties such as mean and variance (e.g., centers and medoids, representative points [24], and CF vectors [69]).

The explicit computation of the local maxima of the density faces two challenges. First, the density maxima of numeric datasets are affected by small density fluctuations. Density fluctuations produce false peaks and a robust identification of local maxima is non-trivial. Second, the explicit computation of maxima of the density quickly becomes prohibitively expensive for large multi-dimensional

datasets. CORE successfully solves these challenges with the help of a multi-dimensional sequential organization of a minimal number of grid points.

Our method does not assume that clusters follow an a-priori model and does not require model parameters that guide the clustering process. CORE is a non-parametric method since the clustering only depends on the precision of the density estimation of the AD-Tree. The precision of the AD-Tree is controlled by ε, which is not a model parameter and exhibits a monotonic behavior for values larger than the kernel bandwidth: a decrease of ε yields an increase of the estimation precision of the AD-Tree and, hence, a higher quality clustering (at the cost of a higher runtime). Related clustering techniques depend on various model parameters like the number of clusters, number of cluster representatives, shrinking parameter, and size of the neighborhood. Model parameters allow to adjust the method to different datasets and application scenarios. This can be useful for specialized applications, but in most cases choosing model parameters significantly complicates the analysis process and there is no guarantee that appropriate model parameters exist to get a specific clustering behavior. In general, the values of model parameters must be computed based on heuristics, statistical properties of the data, domain knowledge, or results of previous clusterings. Choosing model parameters becomes particularly challenging for large multi-dimensional data. In addition, the model parameters exhibit a non-monotonic behavior: an increase or decrease of the values of the parameters does not guarantee an increase of the quality of the clustering.

Problem Definition: Let $D \subset R^d$ be a d-dimensional dataset and let $\hat{f}(x)$ be a continuous estimate of the density function of D. We propose a nonparametric approximation and explicit representation of the maximal areas of $\hat{f}(x)$ that is robust to small density fluctuations.

CORE clusters the data in three steps: (i) computation of the AD-Tree density summary of the data with the help of the kernel density estimation method, (ii) computation of cores in the AD-Tree and assigning grid points to cores, and (iii) labeling the points from the dataset. Figure 3.1 illustrates the steps for a sample two dimensional dataset with two clusters. First, we compute the AD-Tree from a random sample of the dataset. The AD-Tree is a hierarchy of d-dimensional grids (cf. Figure 3.1(a)) that store density values at grid points (cf. Figure 3.1(g)). During the second step we compute the cores and assign grid points to cores. Figure 3.1(c) illustrates cores X_1 and X_2, which are grid points denoted by triangles: X_1 consists of five grid points and X_2 consists of one grid point. Cores X_1 and X_2 approximate two peaks in the density function shown in Figure 3.1(g). The third step labels each tuple with the core of the closest grid point.

The main contributions of our work are the following:

- We develop CORE, a novel clustering technique that explicitly computes density maxima and represents them with cores. CORE does not require any model parameters and ensures a high quality clustering.

- We introduce the rectangular neighborhood of grid points in the AD-Tree, which, together with gradients ensures a robust computation of cores of any dimensionality that is resilient to density fluctuations in the dataset.

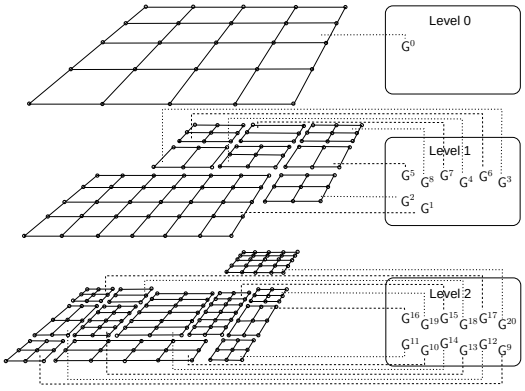
- We extensively evaluate CORE. Our experimental results show that the clustering quality of CORE is superior to state of the art clustering techniques if clusters are close to each other or overlap. CORE efficiently scales up with the dimensionality and the size of the sample.

The chapter is organized as follows. Section 3.2 discusses the related work. Section 3.3 gives preliminaries. Sections 3.4 and Section 3.5 define rectangular neighborhood and cores, respectively. Section 3.6 explains the clustering of grid and data points. Section 3.7 gives analytical investigation of CORE. Section 3.8 presents the clustering algorithms and Section 3.9 evaluates CORE experimentally. Section 3.10 summarizes and concludes the chapter.

3.2 Related Work

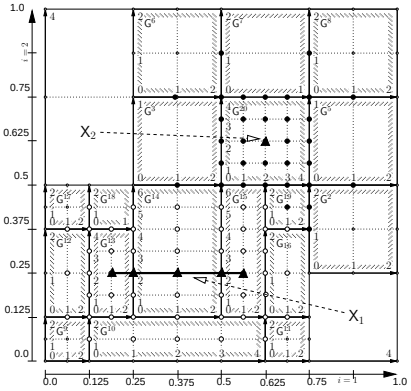
DBScan [17], OPTICS [3], and DENCLUE 2.0 [33] are density-based techniques that compute local maxima implicitly. Clusters identified by DBScan satisfy the following properties: *i*) inside a cluster there are at least $minPts$ points within radius ε and *ii*) border points of clusters are density reachable from points located inside the clusters. OPTICS observes that the identification of clusters at all density levels is not possible with $minPts$ and ε only. OPTICS orders data points according to their reachability distances and computes a reachability plot that summarizes DBScan for different values of ε, but introduces a parameter for reachability plots. Clusters in reachability plots correspond to valleys between steep areas and are easily determined visually. Similarly to DBScan, DENCLUE identifies clusters only at one density level and is sensitive to local fluctuation of the density. It employs two model parameters: σ controls the radius of the neighborhood and ξ defines the lowest density level. DENCLUE assigns data points to peaks of the density by moving them along gradients in automatically computed steps. DENCLUE efficiently captures spherical clusters but does not provide an algorithm for the computation of arbitrarily shaped clusters: according to the definition in Hinneburg et al. [28] grouping density peaks into arbitrarily shaped clusters is an NP complete problem. All techniques model clusters with dense areas and do not model the local maxima explicitly. A careful selection of the input parameters (if possible at all) is required to achieve a good clustering.

Data Bubbles [9] compresses a large dataset into a small number of data bubbles that improve the performance of OPTICS by an order of magnitude. Similar to clustering features in BIRCH, data bubbles store statistical information of a subset of the data. The detailed statistical information maintained by Data



(a) 2-dimensional AD-Tree consists of 3 levels and of 21 regular grids.

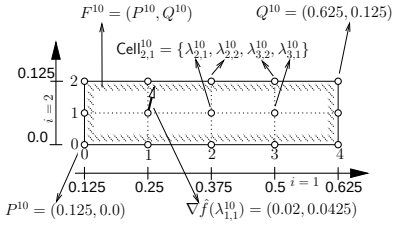
2nd Step: compute cores and assign grid points



(c) Triangular grid points are cores. White grid points are assigned to core X_1. Black grid points are assigned to core X_2. Dots are unassigned grid points (noise).

1st Step: compute AD-Tree from random sample of dataset

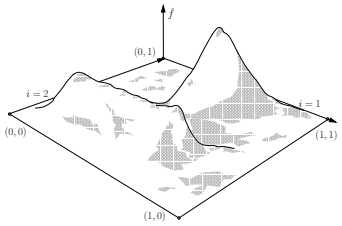
3rd Step: for each tuple t_s query AD-Tree and assign cluster ID of the closest grid point



(e) Grid G^{10} and Involved Notation

	$i=1$	$i=2$	Cluster ID
t_1	0.31	0.32	1
t_2	0.6	0.37	1
t_3	0.61	0.72	2
t_4	0.76	0.77	2
t_5	0.49	0.63	2
t_6	0.38	0.35	1
...	...	...	...
t_s	0.6	0.61	2
...	...	...	...
t_n	0.1	0.83	*noise*

(f) Sample 2-dimensional Dataset



(g) Density Function

Figure 3.1: Three steps of CORE: computation of AD-Tree, clustering of grid points and querying.

Bubbles allows a good clustering quality for high compression rates. Data Bubbles introduces the size of the control sample as a parameter.

Similar to DBScan and OPTICS, CLIQUE [2], WaveCluster [54] and Shrinking [56] are grid based clustering techniques and model clusters by dense areas (union of dense cells). CLIQUE finds all low-dimensional subspaces that contain clusters. WaveCluster uses wavelets to transform the data into the frequency domain where it computes k-connected dense cells. Shrinking moves data points of dense cells towards centroids and efficiently finds condensed and well-separated clusters. The techniques depend on parameter τ, which defines dense cells and parameter ξ, which defines the width of a cell. If the data within the cells are distributed uniformly then the dense cells yield the same results as our technique. If the cells are not distributed uniformly the ξ parameter must be selected very carefully to avoid that a cluster is split. The negative effects of an incorrectly chosen ξ can be partially eliminated with WaveCluster with two other parameters: the radius

ϵ and the number of cells k in the k-ϵ neighborhood of a cell. Shrinking addresses the parameter selection by finding clusters on several differently sized grids and selecting the best quality clusters. The techniques work well for datasets with clusters that can be separated by a single density level. Still, they depend on a good choice of ξ, particularly, if clusters differ in size, shape or distribution. In contrast, CORE adaptively allocates the width ξ of cells and models the clusters with explicit local maxima.

BIRCH [69] and CURE [24] are hierarchical clustering techniques that model clusters with the help of average and standard deviation of clusters (clustering features). BIRCH organizes clustering features, each computed for a subset of the data, into the CF tree. The clustering features of each leaf node represent non-overlapping d-dimensional disks that partition the entire dataset. CURE models clusters with c representative points and the α shrinking factor. Representative points naturally extend the modeling of clusters from one (average) point per cluster to multiple representative points per cluster. In each iteration CURE chooses the most scattered points in the cluster as representative points and shrinks the representative points towards the centroids by factor α to avoid anomalies. For symmetric clusters the average of the cluster coincides with the local maxima. For asymmetric clusters, with multiple maxima points, the quality of the BIRCH and CURE clustering deteriorates. The method could place the average and the representative points even outside the actual clusters. Adjusting the input parameters can help to reduce the negative impact for spherical clusters but remains difficult or impossible for clusters with local maxima forming curves, surfaces, or other non-single point sets.

RIC [7] and ClusteringAggregation [22] are the latest general purpose techniques that aim to improve the quality of existing parametric clustering methods. RIC is an automatic framework to refine clusters by assigning to each of them a probability density function and, then, merging them based on the Volume After Compression (VAC) criterion. ClusteringAggregation proposes five algorithms that aggregate results of several clustering techniques according to the measure of disagreement. Both techniques improve the clustering quality, but cannot split merged or identify undetected clusters. Moreover, our experiments show that RIC improves the quality only if clusters are clearly separated in space. The complexity of the techniques is quadratic in the number of data points and do not scale to large datasets.

Recently, a number of parametric clustering techniques were proposed to deal with specific data. Chen et al. [13] and Cao et al. [10] are new methods for clustering data streams without assumption about the number of clusters. Chen et al. [13] is density based and uses a decay factor to deal with changing clusters in data streams. Cao et al. [10] discover arbitrarily shaped clusters with the help of a core-micro-cluster synopsis. Both, works depend on four input parameters. Kriegel at al. [38, 37] adopt OPTICS and DBScan for clustering uncertain data by representing uncertainty with a distance density function. Zhao at al. [70] pro-

pose a graph-based clustering technique which, based on seven input parameters, finds coherent clusters in three-dimensional gene expressions. Moise at al. [45] propose a parametric subspace clustering technique that is based on statistically significant regions. Network clustering is investigated in Xu et al. [66]. Evolutionary clustering [11] considers the problem of clustering data over time. Binary clustering is investigated in Li [40]. Wu et al. [64] improve K-Means for sparse data by replacing the objective function with the Shannon entropy. 4C [8] and CURLER [59] incorporate linear correlation information into clustering with the help of a λ-dimensional linear set. None of these methods models clusters with explicit local maximas and the methods are based on parametric models with different model parameters.

3.3 Preliminaries

CORE clustering uses the AD-Tree to incrementally compute the density estimate $\hat{f}$ and compress it into a hierarchy of d-dimensional grids. This section illustrates and refreshes definitions in Chapter 2 of the key elements of the AD-Tree: *frame*, *grid*, and *cell*.

Throughout we use the following notation. We denote a *grid* by G^k, a *grid point* by λ^k_J, and a *cell* by C^k_J. $J = (j_1, \ldots, j_d)$ is d-dimensional index to identify cells and grid points within a grid. We denote the gradient at point x by $\nabla\hat{f}(x)$. We write $[x]_i$ to refer to the coordinate of a d-dimensional point x in the i-th dimension. Grid, frame, cell and grid point belong to a node in the AD-Tree. We use superscript ^{k} for the index of the node. The *origin* is the d-dimensional point with coordinate $(0, ..., 0)$.

Definition 3.1. *[Frame] A* frame $F^k = (P^k, Q^k)$ *is a pair of d-dimensional points.* P^k *and* Q^k *define a d-dimensional hyper-rectangle with edges parallel to the coordinate axes.* $P^k = (P^k_1, \ldots, P^k_d)$ *is closest to the origin and* $Q^k = (Q^k_1, \ldots, Q^k_d)$ *is farthest from the origin.*

Definition 3.2. *[Granularity] A* granularity $S^k = (S^k_1, \ldots, S^k_d)$ *is a d-dimensional vector of positive integers.*

Definition 3.3. *[Grid] Let* $F^k = (P^k, Q^k)$ *be a frame and* S^k *be a granularity. A* grid $\mathsf{G}^k(F^k, \mathsf{S}^k) = \{\lambda^k_{J=(j_1,\ldots,j_d)} : j_i = 0, \ldots, \mathsf{S}^k_i; i = 1, \ldots, d\}$ *is a set of grid points where* $[\lambda^k_J]_i = P^k_i + j_i(Q^k_i - P^k_i)/\mathsf{S}^k_i$.

Example 3.1. *[Grid] Consider grid* G^{10} *in Figures 3.1(e) and 3.1(a).* $F^{10} = \big((0.125, 0.0), (0.625, 0.125)\big)$ *is the frame and* $\mathsf{S}^{10} = (4, 2)$ *is the granularity of* G^{10}*. There are* $(\mathsf{S}^{10}_1 + 1) \cdot (\mathsf{S}^{10}_2 + 1) = 15$ *grid points in* G^{10}*. Along each dimension grid points are equally spaced, and in our example the position of grid point* $\lambda^{10}_{3,1}$ *in dimension* $i = 1$ *is* $[\lambda^{10}_{3,1}]_1 = 0.125 + 3 \cdot (0.625 - 0.125)/4 = 0.5$ *and in dimension* $i = 2$ *is* $[\lambda^{10}_{3,1}]_2 = 0.0 + 1 \cdot (0.125 - 0.0)/2 = 0.0625$.

Definition 3.4. *[Cell] Let* $\mathsf{G}^k(F^k, \mathsf{S}^k)$ *be a grid and* $J = (j_1, \ldots, j_d) < (S_1^k, \ldots, S_d^k)$ *be a d-dimensional index. A cell* C_J^k *consists of all vertices on a minimal frame and is identified by the index of the grid point that is closest to the origin:* $\mathsf{C}_J^k = \{\lambda^k_{l_1,\ldots,l_d} : l_i \in \{j_i, j_i + 1\}, i = 1, \ldots, d\}$.

Example 3.2. *[Cell] Consider grid* G^{10} *in Figures 3.1(e) and 3.1(c). Then,* $\mathsf{C}^{10}_{2,1} = \{\lambda^{10}_{2,1}, \lambda^{10}_{2,2}, \lambda^{10}_{3,2}, \lambda^{10}_{3,1}\}$.

Definition 3.5. *[Gradient] The gradient* $\nabla \hat{f}(x)$ *at point* x *is a d-dimensional vector of derivatives of* $\hat{f}$ *at* x*:*

$$\nabla \hat{f}(x) = \Big(\frac{\partial \hat{f}}{\partial x^1}(x), \ldots, \frac{\partial \hat{f}}{\partial x^d}(x)\Big)$$

Example 3.3. *[Gradient] Consider grid* G^{10} *in Figures 3.1(e). The arrow at grid point* $\lambda^{10}_{1,1} = (0.25, 0.0625)$ *is gradient* $\nabla \hat{f}(\lambda^{10}_{1,1}) = (0.02, 0.0425)$.

3.4 Rectangular Neighborhood

A rectangular neighborhood localizes stationary points of $\hat{f}$ in the AD-Tree and is the key concept for the exact computation of core points. A core point is a stationary point that correspond to local maxima. Rectangular neighborhood may span multiple grids and are generalizations of frame and cell, respectively.

Intuitively, a rectangular neighborhood consists of grid points that lie on the sides of an *embedding frame*. The embedding frame $E(\lambda_J^k)$ encloses a grid point λ_J^k into a minimal frame that satisfies the following requirements: *i*) only one grid point λ_J^k lies inside $F(\lambda_J^k)$ and *ii*) each edge of $F(\lambda_J^k)$ contains at least two points of cells that include λ_J^k.

3.4.1 Embedding Frame

Figure 3.2 illustrates the embedding frame $E(\lambda_J^k) = ((0.5625, 0.0625), (0.6875, 0.1875))$ of grid point $\lambda^{16}_{0,0}$. There is only grid point $\lambda^{16}_{0,0}$ inside $E(\lambda_J^k)$ and each edge of $E(\lambda_J^k)$ includes at least two grid points of a cell that include $\lambda^{16}_{0,0}$.

Definition 3.6. *[Embedding Frame]. Let* $\Phi(\lambda_J^k)$ *be all grid points from cells that contain a grid point equal to* λ_J^k*:*

$$\Phi(\lambda_J^k) = \bigcup_{\mathsf{C}_M^l : \lambda_J^k \in \mathsf{C}_M^l} \mathsf{C}_M^l$$

Let $\Phi_i^-(\lambda_J^k)$ *be the set of ith coordinates smaller than the ith coordinate of* λ_J^k*:*

$$\Phi_i^-(\lambda_J^k) = \big\{ [\lambda]_i : [\lambda]_i < [\lambda_J^k]_i, \lambda \in \Phi(\lambda_J^k) \big\} \tag{3.1}$$

$$\Phi_i^+(\lambda_J^k) = \big\{ [\lambda]_i : [\lambda]_i > [\lambda_J^k]_i, \lambda \in \Phi(\lambda_J^k) \big\} \tag{3.2}$$

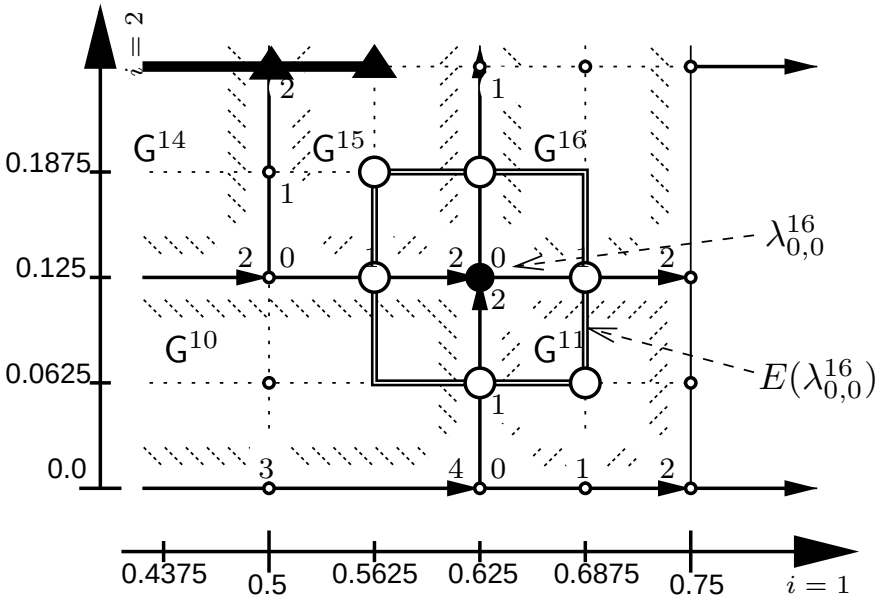


Figure 3.2: Incomplete Rectangular Neighborhood

The embedding frame $E(\lambda_J^k) = (P(\lambda_J^k), Q(\lambda_J^k))$ *for* λ_J^k *is defined for each coordinate as follows:*

$$\big[P(\lambda_J^k)\big]_i = \begin{cases} \max\{\Phi_i^-(\lambda_J^k)\} & \text{if } \Phi_i^-(\lambda_J^k) \neq \emptyset \\ \big[\lambda_J^k\big]_i & \text{otherwise} \end{cases} \tag{3.3}$$

$$\big[Q(\lambda_J^k)\big]_i = \begin{cases} \min\{\Phi_i^+(\lambda_J^k)\} & \text{if } \Phi_i^+(\lambda_J^k) \neq \emptyset \\ \big[\lambda_J^k\big]_i & \text{otherwise.} \end{cases} \tag{3.4}$$

Example 3.4. *[Embedding Frame]. Consider* $\lambda_{0,0}^{16} = (0.625, 0.125)$ *in Figure 3.2. The set* $\Phi(\lambda_{0,0}^{16})$ *of grid points of cells that contain grid point* $\lambda_{0,0}^{16}$ *is*

$$\begin{aligned} \Phi(\lambda_{0,0}^{16}) = \big\{&(0.5, 0.0625), (0.625, 0.0625), (0.5, 0.125), \\ &(0.5625, 0.125), (0.5625, 0.1875), (0.625, 0.1875), \\ &(0.6875, 0.0625), (0.6875, 0.125), (0.625, 0.25), \\ &(0.6875, 0.25)\big\} \end{aligned}$$

Set $\Phi_1^-(\lambda_{0,0}^{16})$ *contains the first coordinates of* $\Phi(\lambda_{0,0}^{16})$ *that are smaller than the first coordinate of* $\lambda_{0,0}^{16}$*. Similarly,* $\Phi_2^-(\lambda_{0,0}^{16})$ *contains the second coordinates of* $\Phi(\lambda_{0,0}^{16})$ *that are smaller than the second coordinate of* $\lambda_{0,0}^{16}$*:*

$$\begin{aligned} \Phi_1^-(\lambda_{0,0}^{16}) &= \big\{0.5, 0.5625\big\} \\ \Phi_2^-(\lambda_{0,0}^{16}) &= \big\{0.0625\big\} \end{aligned}$$

From Equation 3.3:

$$\begin{aligned} P(\lambda_{0,0}^{16}) &= \Big(\max\big\{\Phi_1^-(\lambda_{0,0}^{16})\big\}, \max\big\{\Phi_1^-(\lambda_{0,0}^{16})\big\}\Big) \\ &= \big(0.5625, 0.0625\big). \end{aligned}$$

The computation of $Q(\lambda_{0,0}^{16})$ *is equivalent. Thus, the embedding frame of* $\lambda_{0,0}^{16}$ *is* $E(\lambda_{0,0}^{16}) = \big(P(\lambda_{0,0}^{16}), Q(\lambda_{0,0}^{16})\big) = \big((0.5625, 0.0625), (0.6875,\ 0.1875)\big)$.

3.4.2 Rectangular Neighborhood

The white circles in Figures 3.2 and 3.3 illustrate the rectangular neighborhoods for grid points $\lambda_{0,0}^{16}$ and $\lambda_{1,1}^{11}$. Intuitively, the rectangular neighborhood of a grid point λ_J^k consists of grid points that are on the embedding frame of λ_J^k and belong to the set $\Phi(\lambda_J^k)$.

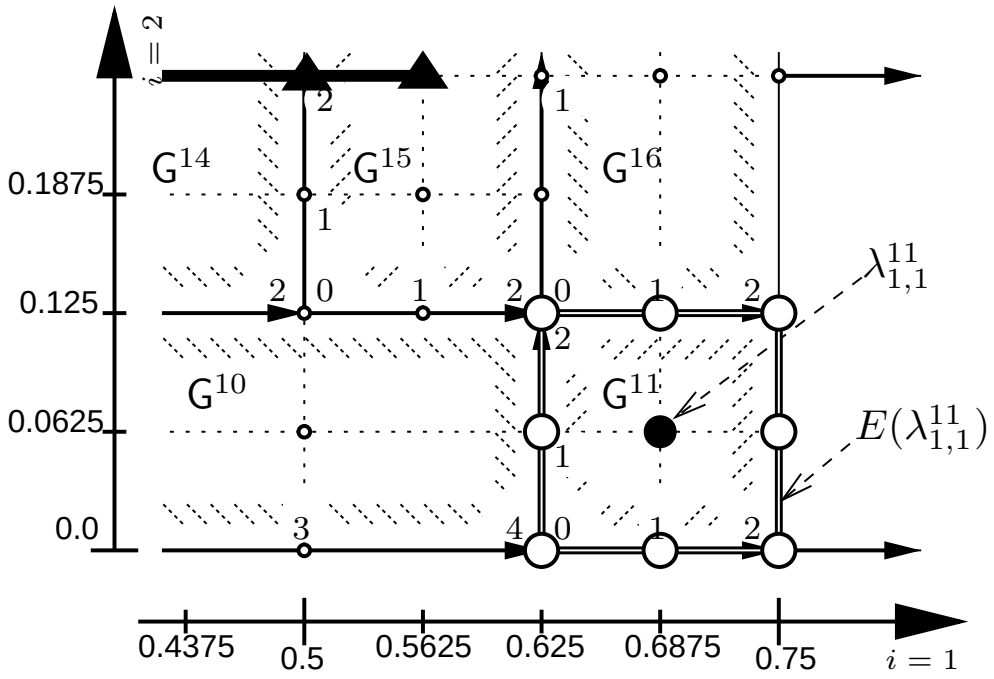


Figure 3.3: Complete Rectangular Neighborhood

Definition 3.7. *[Rectangular Neighborhood]. Let* $E(\lambda_J^k) = \big(P(\lambda_J^k), Q(\lambda_J^k)\big)$ *be the embedding frame of grid point* λ_J^k*. The rectangular neighborhood of grid point* λ_J^k *consists of all grid points in* $\Phi(\lambda_J^k)$ *that are on the embedding frame:*

$$\begin{aligned} \mathsf{R}(\lambda_J^k) = \Big\{ & \lambda \in \Phi(\lambda_J^k) : \\ & [P(\lambda_J^k)]_i \le [\lambda]_i \le [Q(\lambda_J^k)]_i, i = 1, \ldots, d \Big\} \backslash \lambda_J^k \end{aligned}$$

where $\Phi(\lambda_J^k)$ *is defined according to Definition 3.6.*

Example 3.5. *[Rectangular Neighborhood]. Rectangular neighborhood of* $\lambda_{0,0}^{16}$ *is the following set*

$$\mathsf{R}(\lambda_{0,0}^{16}) = \{(0.5625, 0.1875), (0.625, 0.1875), (0.5625, 0.125), (0.625, 0.125), (0.6875, 0.125), (0.625, 0.0625), (0.6875, 0.0625)\}$$

The rectangular neighborhood $\mathsf{R}(\lambda_J^k)$ is maximal if all grid points of $\Phi(\lambda_J^k) \setminus \{\lambda_J^k\}$ are on the embedding frame. We call such a rectangular neighborhood complete. In Figure 3.3, all grid points of cells that contain $\lambda_{1,1}^{11}$, except $\lambda_{1,1}^{11}$ are on the embedding frame and therefore $\mathsf{R}(\lambda_{1,1}^{11})$ is complete.

Definition 3.8. *[Complete and Incomplete Rectangular Neighborhood]. Rectangular neighborhood* $\mathsf{R}(\lambda_J^k)$ *is complete if* $|\mathsf{R}(\lambda_J^k)| = 3^d - 1$ *and incomplete otherwise.*

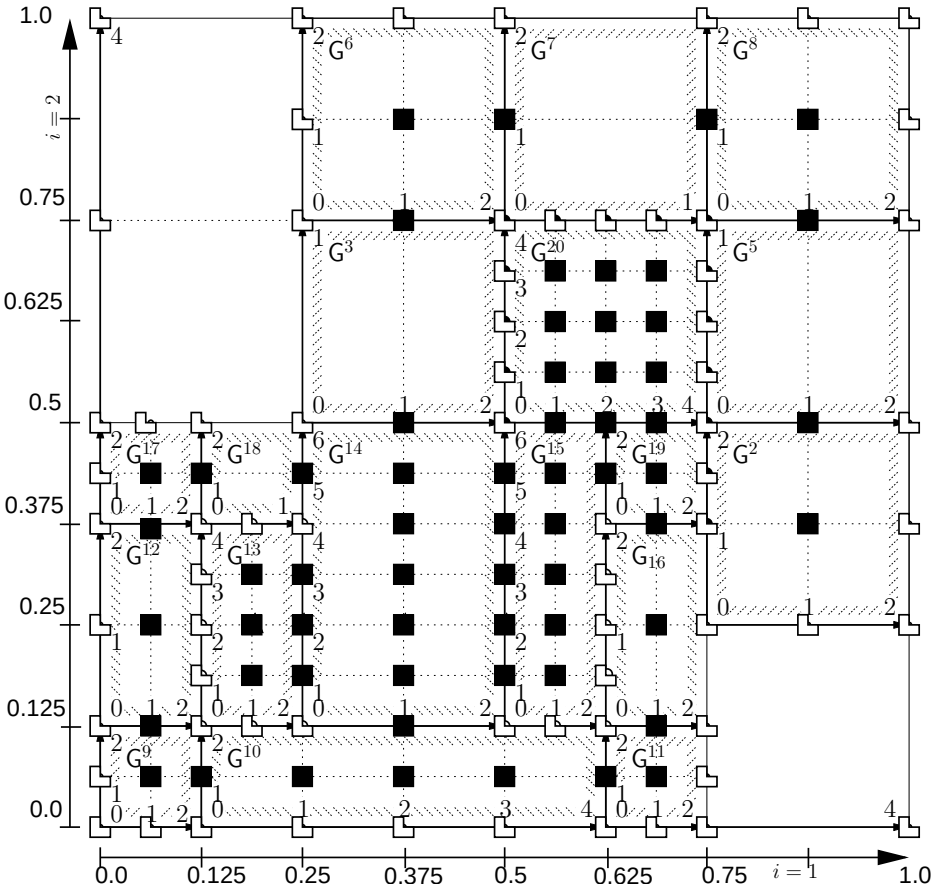


Figure 3.4: Grid Points with Complete and Incomplete Rectangular Neighborhood.

	1st Condition	2nd Condition
Core Point	Satisfied	Satisfied
Neighborhood Point	Satisfied	Unsatisfied
Saddle Point	Unsatisfied	Satisfied
Inflection Point	Unsatisfied	Unsatisfied
Minima	Unsatisfied	Unsatisfied
Not Stationary Point	Unsatisfied	Unsatisfied

Table 3.1: Definition 3.9 for Different Points.

In Figure 3.4 black squares are grid points with a complete rectangular neighborhood and white polygons are grid points with an incomplete rectangular neighborhood. White polygons occurs only in places where $\hat{f}$ is monotonically increasing or decreasing, i.e., in places where $\hat{f}$ has no stationary points.

Theorem 3.1. *A grid point with an incomplete rectangular neighborhood is not a stationary point of $\hat{f}$.*

3.5 Cores of Clusters

A Core approximates a set of local maxima points of the density function of the data from the AD-Tree. Rectangular neighborhood efficiently localizes stationary points: places where the derivative of the density function is zero, including local minima, local maxima, saddle and inflection points. In the following definition we formulate specific conditions that captures only the local maxima and leaves out all other types of stationary points.

Definition 3.9. *[Core of a Cluster]. Let $\mathsf{X} = \{\lambda_{J_1}^{k_1}, \ldots, \lambda_{J_t}^{k_t}, \ldots\}$ in a d-dimensional AD-Tree. Let for each $\lambda_{J_t}^{k_t} \in \mathsf{X}$ its rectangular neighborhood $\mathsf{R}(\lambda_{J_t}^{k_t})$ be complete. Then, X is a core of a cluster iff it is a maximal and connected set with the following conditions satisfied:*

1. *Many gradients of points from rectangular neighborhood point towards the core point:*

$$\left|\left\{\lambda \in \mathsf{R}(\lambda_{J_t}^{k_t}) : \angle\left(\lambda_{J_t}^{k_t} - \lambda, \nabla\hat{f}(\lambda)\right) < 90^\circ\right\}\right| \geq 2 \cdot 3^{d-1} \tag{3.5}$$

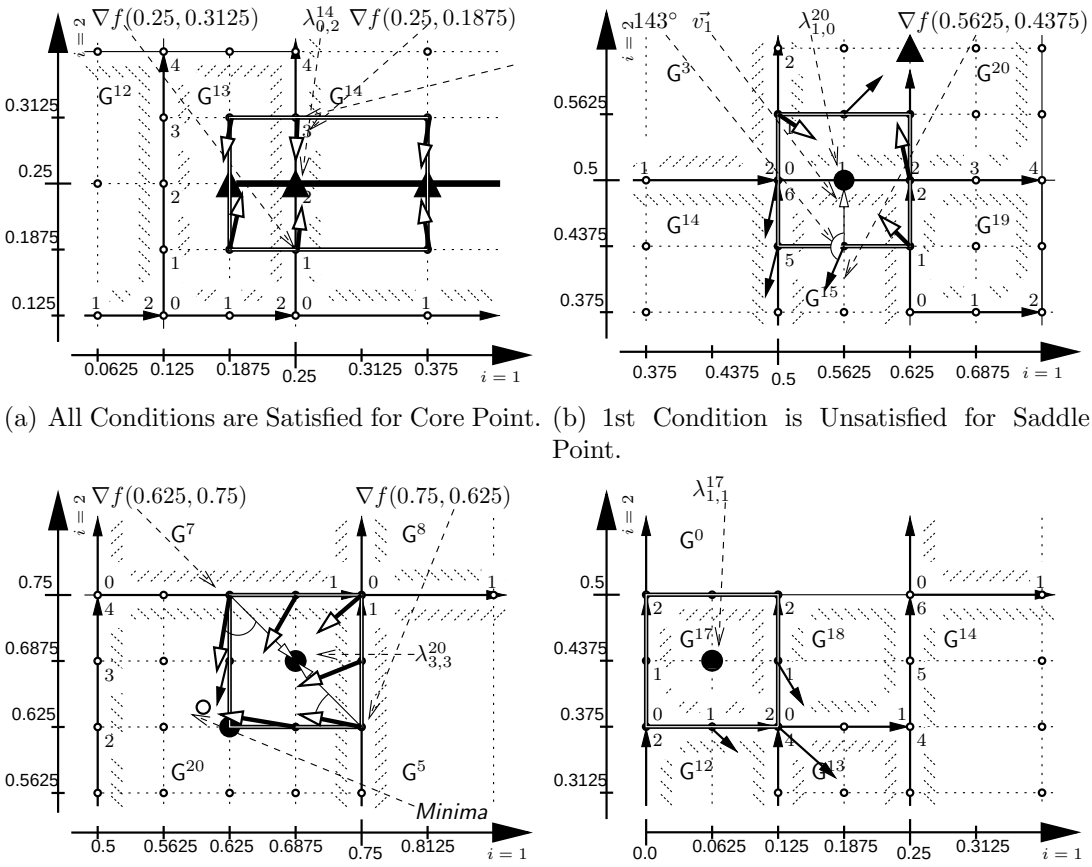
2. *There are two gradients pointing to $\lambda_{J_t}^{k_t}$ from opposite directions: there exists $\lambda, \mu \in \mathsf{R}(\lambda_{J_t}^{k_t})$ such that the following conditions are satisfied:*

   (a) *For each coordinate $i : 1 \leq i \leq d$ one of the above Equation 3.6 and Equation 3.7 is satisfied:*

$$\left[\lambda\right]_i = \left[\mu\right]_i = \left[\lambda_{J_t}^{k_t}\right]_i \tag{3.6}$$

$$\big[\lambda\big]_i = \big[P(\lambda_{J_t}^{k_t})\big]_i \text{ and } \big[\mu\big]_i = \big[Q(\lambda_{J_t}^{k_t})\big]_i \tag{3.7}$$

(b) $\angle\big(\lambda_{J_t}^{k_t} - \lambda, \nabla\hat{f}(\lambda)\big) < 45°$ *and* $\angle\big(\lambda_{J_t}^{k_t} - \mu, \nabla\hat{f}(\mu)\big) < 45°$



(a) All Conditions are Satisfied for Core Point.

(b) 1st Condition is Unsatisfied for Saddle Point.

(c) 2nd Condition is Unsatisfied for Neighbor .

(d) All Condition are Unsatisfied for Minima Points.

Figure 3.5: Comparison of Definition 3.9 for different type of stationary points

Figure 3.5 illustrates how Definition 3.9 distinguishes core points from other types of stationary points. Grid point $\lambda_{0,2}^{14}$ in Figure 3.5(a) is a core point since it satisfies both conditions: 6 gradients (cf. Equation 3.5, $2 \cdot 3^{2-1} = 6$) are pointing towards the core point and there are 2 gradients at $\lambda = \lambda_{0,3}^{14}$ and $\lambda = \lambda_{0,1}^{14}$ pointing to the core point from the opposite directions. Not core points $\lambda_{1,0}^{20}$ in Figure 3.5(b), $\lambda_{3,3}^{20}$ in Figure 3.5(c) and Figure 3.5(d) do not satisfy one or both conditions in Definition 3.9. $\lambda_{1,0}^{20}$ does not satisfy first condition, since, there are only 3 gradients pointing towards the grid point. In fact, $\lambda_{1,0}^{20}$ is a saddle point (maxima point towards one diagonal and minima point towards the other diagonal). $\lambda_{3,3}^{20}$ does not satisfy second condition, since there are no 2 gradients that point to $\lambda_{3,3}^{20}$ from the opposite directions. This situation happens close to the core point (cf. the discussion below). Grid point $\lambda_{1,1}^{17}$ has no gradients in its

neighborhood pointing towards $\lambda_{1,1}^{17}$ and, therefore, is not a core point. In fact, $\lambda_{1,1}^{17}$ is a local minima.

The rationale for at least $2 \cdot 3^{d-1}$ gradients pointing towards core points is the following. If λ_J^k is a core point then all gradients in the rectangular neighborhood of λ_J^k should point towards λ_J^k, except the gradients at grid points of the rectangular neighborhood which also are core points. In general, in d-dimensional datasets with up to (d-1)-dimensional cores (0-d are points, 1-d are curves, ...), there are 3^d number of gradients in the complete rectangular neighborhood of a grid point, and of these 3^{d-1} do not necessarily point towards the core point. Therefore, λ_J^k is a core point iff there are $3^d - 3^{d-1} = 2 \cdot 3^{d-1}$ number of gradients in its rectangular neighborhood that point towards λ_J^k.

Example 3.6. *[Core, first condition, Equation 3.5]. We show that gradient* $\nabla \hat{f}(0.5625, 0.4375)$ *in Figure 3.5(b) is not pointing towards grid point* $\lambda_{1,0}^{20} = (0.5625, 0.5)$.

We compute the angle between the gradient $\nabla \hat{f}(0.5625, 0.4375)$ *and direction connecting points* $(0.5625, 0.4375)$ *and* $(0.5625, 0.5)$. *The angle between any two vectors* $\vec{v_1}$ *and* $\vec{v_2}$ *is computed as following:*

$$\angle(\vec{v_1}, \vec{v_2}) = \arccos\left(\frac{\vec{v_1} \cdot \vec{v_2}}{||\vec{v_1}|| \cdot ||\vec{v_2}||}\right)$$

In our case

$$\vec{v_1} = \nabla \hat{f}(0.5625, 0.4375) = (-0.03, -0.004)$$
$$\vec{v_2} = (0.5625, 0.5) - (0.5625, 0.4375) = (0.0, 0.0625)$$

Therefore,

$$\arccos\left(\frac{(0.0, 0.0625) \cdot (-0.03, -0.004)}{||(0.0, 0.0625)|| \cdot ||(-0.03, -0.004)||}\right) \approx 143^\circ.$$

$143^\circ > 90^\circ$ *and, hence gradient* $\nabla \hat{f}(0.5625, 0.4375)$ *is not pointing towards* $\lambda_{1,0}^{20}$.

Close to the local maxima many gradients are pointing towards one point and, hence, first condition of Definition 3.9 is satisfied for several grid points (cf. Figure 3.5(c)). Second condition ensures that only grid point which is the closest to local maxima is declared as a core point. Grid points $\lambda_{3,3}^{20}$ and $\lambda_{2,2}^{20}$ are approximating the same local maxima and both satisfy first condition. Second condition of Definition 3.9 selects $\lambda_{3,3}^{20}$ as a core point, since, it is the best approximation of the local maxima.

Technically, second condition consists of two sub-conditions (a) and (b): (a) defines opposite points of $\mathsf{R}(\lambda_J^k)$ and sub-condition (b) defines gradients which are pointing to the core point from opposite points of $\mathsf{R}(\lambda_J^k)$. Next, we explain and illustrate them individually.

Example 3.7. *[Core, second condition]. Grid points* $\lambda = (0.625, 0.75)$ *and* $\mu = (0.75, 0.625)$ *from* $\mathsf{R}(\lambda^{20}_{3,3})$ *in Figure 3.5(c) are opposite. Indeed, the embedding frame of* $\lambda^{20}_{3,3}$ *is* $E(\lambda^{20}_{3,3}) = \big((0.625, 0.625), (0.75, 0.75)\big)$. λ *and* μ *are opposite since for each coordinate* i *they satisfy Equation 3.7:*

$$\begin{aligned}\big[\lambda\big]_1 &= 0.625 = \big[P(\lambda^{20}_{3,3})\big]_1\\ \big[\mu\big]_1 &= 0.755 = \big[Q(\lambda^{20}_{3,3})\big]_1\\ \big[\lambda\big]_2 &= 0.750 = \big[P(\lambda^{20}_{3,3})\big]_2\\ \big[\mu\big]_2 &= 0.625 = \big[Q(\lambda^{20}_{3,3})\big]_2\end{aligned}$$

Grid points $\lambda = (0.6875, 0.7500)$ *and* $\mu = (0.7500, 0.6250)$ *of the same* $\mathsf{R}(\lambda^{20}_{3,3})$ *are not opposite since Equation 3.6 and 3.7 are not satisfied for coordinate* $i = 1$*:*

$$\big[\lambda\big]_1 \neq \big[\mu\big]_1; \big[\lambda\big]_1 \neq \big[P(\lambda^{20}_{3,3})\big]_1; \big[\lambda\big]_1 \neq \big[Q(\lambda^{20}_{3,3})\big]_1.$$

The angles between the gradients at the opposite points and the direction to the core point should be less than $45°$. *Any two opposite grid points* $\lambda, \mu \in \mathsf{R}(\lambda^{20}_{3,3})$ *do not satisfy this condition (cf. Figure 3.5(c)) and, hence* $\lambda^{20}_{3,3}$ *is not a core point.Core point* $\lambda^{14}_{0,2}$ *(cf. Figure 3.5(a))satisfies this condition with* $\lambda = (0.25, 0.1875)$ *and* $\mu = (0.25, 0.3125)$.

Summarizing, Table 3.1 compares first and second conditions for different types of points. Only core points satisfy all conditions of Definition 3.9.

3.6 Labeling Grid and Data Points

We label each data point t_s with the cluster number of its closest grid point. The cluster number of grid point λ^k_J is the index of a core to which the gradient path of λ^k_J leads. The gradient path of λ^k_J is a sequence of grid points that starts at λ^k_J and the gradient at each grid point except the last one is pointing towards the next grid point of the path.

Definition 3.10. *[Gradient Path]. The gradient path of* λ^k_J *is a sequence of grid points* $P(\lambda^k_J) = (\lambda_1, \ldots, \lambda_s)$*, such that the first element is* $\lambda_1 = \lambda^k_J$*, last element is a core point and each* λ_t *is a neighbor of* λ_{t-1} *that satisfies:*

$$\begin{aligned}\angle\big(\nabla \hat{f}(\lambda_{t-1}), \lambda_t - \lambda_{t-1}\big) = \\ \min_{x_u \in \mathsf{R}(\lambda_{t-1})} \angle\big(\nabla \hat{f}(\lambda_{t-1}), x_u - \lambda_{t-1}\big)\end{aligned} \tag{3.8}$$

3.7 Analytical Investigation

This section analytically evaluates first condition of Definition 3.9.

Theorem 3.2. *Let L be a d-dimensional line segment satisfying the following two properties: (i) L is a local maxima set of the density function of the data and (ii) L is parallel to one of the axis of the coordinate system. Let $\lambda \in L$, $(P(\lambda), Q(\lambda))$ be the embedding frame such that $P(\lambda)$ and $Q(\lambda)$ are close enough to λ. Then there are at least $2 \cdot 3^{d-1}$ gradients at points from $\Psi(\lambda)$ pointing towards λ.*

Proof. λ is the local maxima point of the density function. Therefore, the function is increasing towards λ close around λ except line L. Let us assume that frame $(P(\lambda), Q(\lambda))$ are selected close enough so the density function is increasing towards the local maxima in the frame. Then all gradients from $\Psi(\lambda) \backslash L$ are pointing towards λ. $|\Psi(\lambda) \backslash L| \geq 2 \cdot 3^{d-1}$ finishes the proof.

Corollary 3.1. *Let L be a hyper rectangle satisfying the following three properties: (i) L is a local maxima set of the density function of the data (ii) L the edges of L are parallel to one of the axis of the coordinate system, and (iii) at least one edge of L is zero. Let $\lambda \in L$, $(P(\lambda), Q(\lambda))$ be the embedding frame such that $P(\lambda)$ and $Q(\lambda)$ are close enough to λ. Then there are at least $2 \cdot 3^{d-1}$ gradients at points from $\Psi(\lambda)$ pointing towards λ.*

The condition that the edges of the core must by parallel to the axis of the coordinate system in Corollary 3.1 can be dropped without the loss of generality. In this case frame $(P(\lambda), Q(\lambda))$ must be selected closer to λ so the density function is increasing towards the local maxima in the frame.

Theorem 3.3. *Let AD-Tree such that the density function in each unsplit cell can be approximated linearly with ε precision. Let λ by a local maxima a grid point of the density function with full rectangular neighborhood $\mathsf{R}(\lambda)$. Then there are at least $2 \cdot 3^{d-1}$ number of gradients at grid points from $\mathsf{R}(\lambda)$ pointing towards λ.*

Proof. The proof follows from Corollary 3.1 and the approximation properties of the AD-Tree .

3.8 Algorithms and Complexity

Algorithms 6 and 7 perform CORE clustering on a sample of input dataset D and have a linear time complexity wrt to the number of grid points and the size of the sample. computeCORE first computes the AD-Tree from a sample of the dataset. The computation of the AD-Tree requires one scan through the sample per each level. Line 3 iterates through all grid points and computes for each grid point the last grid point of the gradient path. Lines 4-6 query the AD-Tree for each data point and find its closest grid point. A data point t_s is labeled with the core to which the gradient path starting at the grid point closest to t_s leads.

Algorithm 6: computeCORE(D)

```
1 compute AD-Tree from sample of D;
2 for λ_J^k of AD-Tree do set λ_J^k.processed = false;
3 CORES = ∅;
4 for λ_J^k of AD-Tree do  λ_J^k.last = lastOfP(λ_J^k);
5 for t_s ∈ D do
6     Set t_s.ClusterId = i : λ_J^k.last ∈ X_i, where λ_J^k is the closest to t_s.
7 end
```

lastOfP recursively computes and returns the last grid point of the gradient path that starts at a given grid point λ_J^k. First, at line 3, lastOfP computes the rectangular neighborhood of λ_J^k. Next, if gradients of $\mathsf{R}(\lambda_J^k)$ satisfy definition 3.9, then λ_J^k is a core point and, hence, the last element of the gradient path. Otherwise, lines 8-11 compute the next grid point of the path and repeat lastOfP. The first line of lastOfP ensures that each grid point is considered only once. Lines 4-7 add a new core point to a core and refine all computed cores so that they are maximal and connected.

Algorithm 7: lastOfP(λ_J^k)

```
1  if λ_J^k.processed then return λ_J^k.last;
2  λ_J^k.processed = true;
3  compute R(λ_J^k) by querying AD-Tree;
4  if R(λ_J^k) satisfies Definition 3.9 then
5      CORES = CORES ∪ {λ_J^k};
6      merge all X_1, X_2 ∈ CORES that satisfy X_1 ∩ (R(λ_J^k) ∪ λ_J^k) ≠ ∅ and
       X_2 ∩ (R(λ_J^k) ∪ λ_J^k) ≠ ∅;
7      λ_J^k.last = λ_J^k;
8  else
9      find λ ∈ R(λ_J^k) satisfying Equation 3.8;
10     λ_J^k.last = lastOfP(λ);
11 end
12 return λ_J^k.last;
```

The computation of the rectangular neighborhood of λ_J^k is done with a single query on the AD-Tree. We start with the root grid and initialize $\Phi(\lambda_J^k)$ (cf. Definition 3.6) with all cells of the root that contain λ_J^k. Next, we refine $\Phi(\lambda_J^k)$ by descending the AD-Tree. We scan child grids and check if a child partitions a cell in $\Phi(\lambda_J^k)$. If so, then we substitute partitioned cell with cells from the child grid that contain λ_J^k. When the leaf level is reached, we compute the embedding frame and the rectangular neighborhood from $\Phi(\lambda_J^k)$ following Definition 3.6 and Definition 3.7. In the worst case, querying the AD-Tree checks all grids, in the

best case one grid is checked. Typically, the number of processed grids is close to the number of levels.

3.9 Experiments

We compare CORE with CURE, K-Means, DBScan, OPTICS, DataBubbles, and Robust Information-Theoretic Clustering (RIC) which are efficient for a broad range of datasets. We run these techniques for many different input parameters and present only their best results. We implement THE automatic computation of clusters in OPTICS based on the hierarchy of steep down and steep up regions of reachability plots [3]. Independent of the dataset we build the AD-Tree with a fixed $\varepsilon = 0.02$.

We measure the clustering quality in terms of *precision*, *recall* and *F-score*. Let $W \subseteq D$ be a cluster in the dataset and let $V \subseteq D$ be a computed cluster. Then we define precision, recall and F-score of V wrt to W as follows:

precision $p(V,W) = |V \cap W|/|V|$

recall $r(V,W) = |V \cap W|/|W|$

F-score $F(V,W) = 2 \cdot p(V,W) \cdot r(V,W)/(p(V,W) + r(V,W))$

Let $\mathcal{W}$ be the set of all clusters in a dataset and let $\mathcal{V}$ be the set of computed clusters. We compute the matching from $\mathcal{W}$ to $\mathcal{V}$ that maximizes the average of the F-scores of each pair $(V \in \mathcal{V}, W \in \mathcal{W})$ in the matching.

3.9.1 Quality of Clustering

Table 3.2 compares CORE for four synthetic datasets. The datasets differ in the number, position and type of clusters. Up to 5% of the data points in each dataset are noise. The F-score shows that CORE performs significantly better than other techniques. CORE clearly outperforms the other methods for databases with complex shaped clusters (cf. TwoSpirals in first row) and hierarchical clusters (cf. HierarchicalClusters in second row). For overlapping clusters (cf. Overlapping-Spheres in third row) CORE has a very high average F-score (95.7), but also the other solutions perform fairly well (76.5-83.2). For each dataset we used all its data points and, hence, quality of DataBubbles correspond to quality of OPTICS in Table 3.2.

For the TwoSpirals dataset, CORE correctly identifies both spirals and has the highest average F-score. OPTICS, CURE and DBScan accurately identify the inner spiral, even outperforming CORE a bit (cf. OPTICS with F-score of 99.6 vs. CORE with 97.1). However, all competitors are substantially worse (by 36.6–86.1) for the outer spiral. CURE exhibits a low F-score value, since it shifts

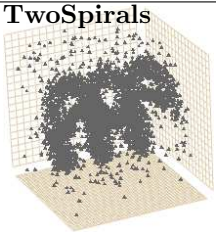
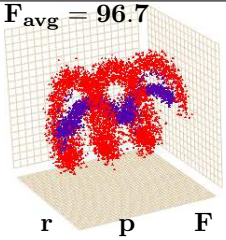
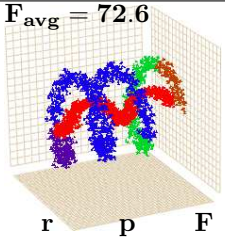
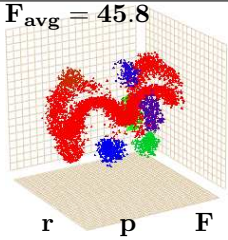
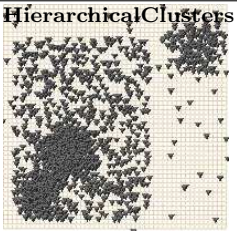
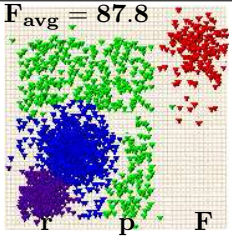
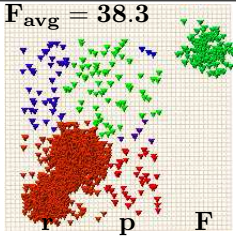
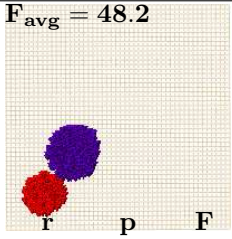
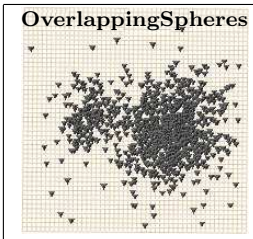
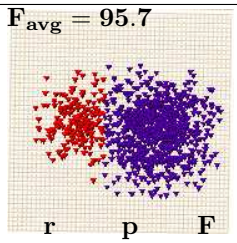
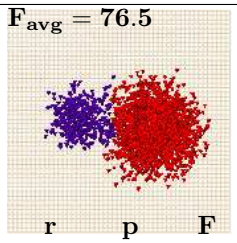
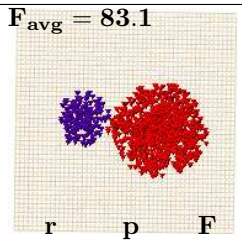
Dataset	CORE			CURE			DBScan		
TwoSpirals 	$\mathbf{F_{avg} = 96.7}$ 			$\mathbf{F_{avg} = 72.6}$ 			$\mathbf{F_{avg} = 45.8}$ 		
	r	**p**	**F**	**r**	**p**	**F**	**r**	**p**	**F**
Inner Spiral	100.0	94.27	97.1	91.8	99.9	95.7	100.0	68.5	81.3
Outer Spiral	92.88	100.0	96.3	32.9	100.0	49.5	5.4	100.0	10.2
HierarchicalClusters 	$\mathbf{F_{avg} = 87.8}$ 			$\mathbf{F_{avg} = 38.3}$ 			$\mathbf{F_{avg} = 48.2}$ 		
	r	**p**	**F**	**r**	**p**	**F**	**r**	**p**	**F**
Embedded Sphere 1	99.1	86.9	92.5	0.0	0.0	0.0	95.3	100.0	97.6
Embedded Sphere 2	98.8	80.8	88.9	95.0	53.7	68.6	90.6	100.0	95.0
Outside Sphere	99.9	99.7	99.8	55.0	100.0	70.9	0.0	0.0	0.0
Plane	56.8	90.8	69.9	7.3	97.1	13.6	0.0	0.0	0.0
OverlappingSpheres 	$\mathbf{F_{avg} = 95.7}$ 			$\mathbf{F_{avg} = 76.5}$ 			$\mathbf{F_{avg} = 83.1}$ 		
	r	**p**	**F**	**r**	**p**	**F**	**r**	**p**	**F**
Right Sphere	97.9	98.6	98.2	72.3	99.6	83.8	80.3	98.7	88.6
Left Sphere	94.4	92.0	93.2	57.6	86.6	69.2	65.7	95.0	77.7
Thirty Spheres	99.0			98.5			97.1		

Table 3.2: Numerical and Visual Comparison of Clustering Quality for Different Distribution of Clusters

the control points toward the inner spiral and the places where the spirals are close to each other. OPTICS and DBScan fail because valleys in the reachability plots are not separated by a steep area. The best clustering for OPTICS and CURE is achieved if the number of clusters is set to five. This leads to one cluster being assigned to the inner spiral and the other four clusters to fragments of the outer spiral. Higher k values splits the inner spiral, while lower k values merge the inner with the outer spiral. The best clustering of DBScan merges the inner spiral with the outer spiral. Adjusting the other parameters does not improve the quality: it either eliminates the outer spiral as noise or merges it with the inner spiral. The best output of K-Means is two clusters having approximately the same number of data points and dividing each spiral in two equal parts. For more than 15

clusters K-Means does not merge the spirals but represents each with a number of spherical clusters. For any output of K-Means, RIC merges all parts of spirals into one cluster that results on the lowest F-score. RIC fails to separate spirals because of two reasons: *i*) spirals cannot be modeled by one probability density function as done in RIC; *ii*) RIC considers only the global orientation of clusters which is similar for both spirals.

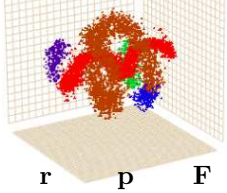
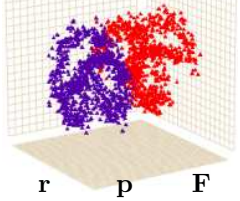
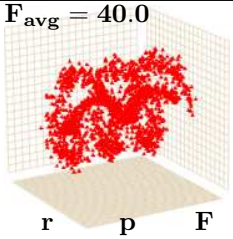
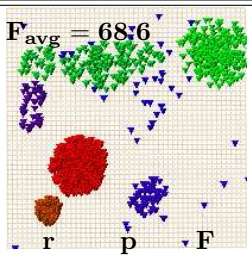
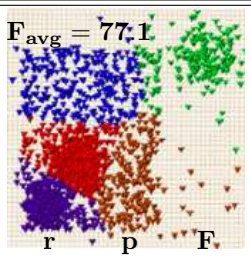
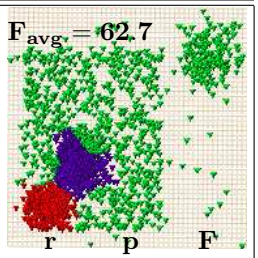
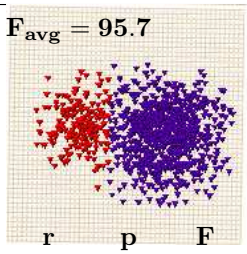
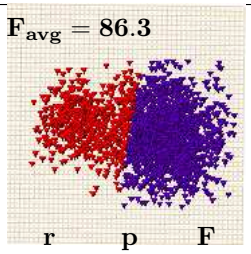
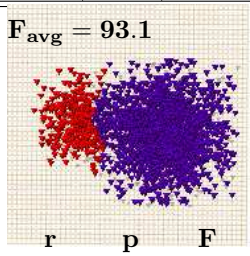
	OPTICS and DataBubbles			K-Means			RIC		
Two Spirals	$\mathbf{F_{avg} = 79.7}$ 			$\mathbf{F_{avg} = 48.6}$ 			$\mathbf{F_{avg} = 40.0}$ 		
	99.7	99.5	99.6	50.0	67.0	57.2	100.0	66.8	80.15
				50.3	33.2	40.0	-	-	-
Hierarchical Clusters	$\mathbf{F_{avg} = 68.6}$ 			$\mathbf{F_{avg} = 77.1}$ 			$\mathbf{F_{avg} = 62.7}$ 		
	95.4	88.3	91.7	95.8	78.8	86.5	86.4	86.3	86.3
	53.4	100.0	69.6	100.0	73.8	84.9	98.4	83.8	90.5
	86.9	100.0	93.0	100.0	75.6	86.1	-	-	-
	11.2	94.5	20.0	35.29	92.1	51.0	98.9	59.6	74.3
Overlapping Spheres	$\mathbf{F_{avg} = 95.7}$ 			$\mathbf{F_{avg} = 86.3}$ 			$\mathbf{F_{avg} = 93.1}$ 		
	80.3	98.7	88.6	87.2	100.0	93.2	99.5	95.5	97.5
	65.9	95.0	77.8	100.0	66.0	79.5	81.4	97.6	88.7
	96.7			90.7			22.4		

Table 3.2: Numerical and Visual Comparison of Clustering Quality for Different Distribution of Clusters. Continued From Previous Page

The HierarchicalClusters dataset is the most challenging for all clustering techniques. The dataset consists of a plane, two dense spheres embedded in the plane cluster, and a sparse sphere outside the plane. All competitors perform poorly. CURE merges half of the plane and the embedded spheres into one

cluster. It splits the other half of the plane and removes many border points of the outside sphere. DBScan separates clusters according to the density level and is able to identify the embedded spheres only. In contrast to DBScan, OPTICS is able to analyze all density levels, but due to the fluctuations in the reachability plot OPTICS splits the plane and removes many border points from the spheres. K-Means finds all spheres however fails with the plane: the upper and right parts of a the plane are identified with two clusters and other parts are assigned to the spheres. RIC separates dense spheres from the plane, however, merges sparse outside sphere with the plane due to the similarity in their density distribution.

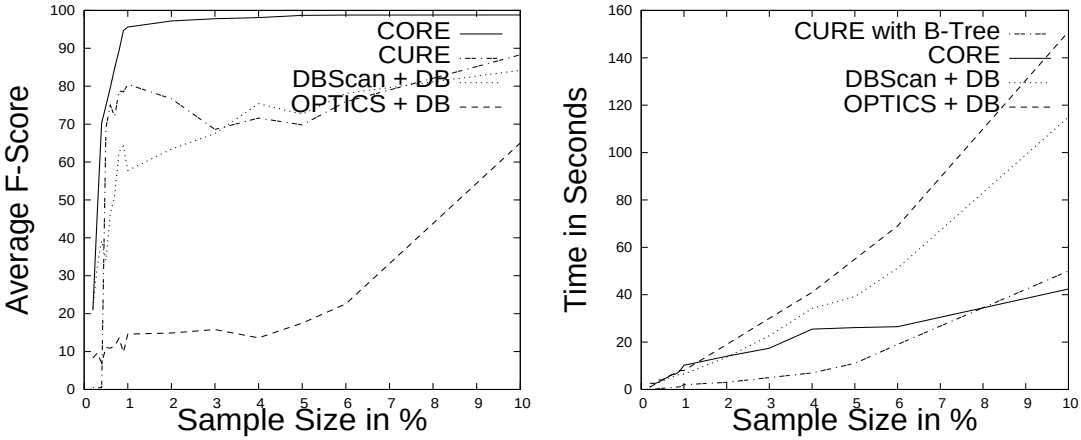
The OverlappingSpheres dataset compares techniques for two overlapping spherical clusters. CORE is the most accurate. CURE incorrectly assigns some points of the left sphere to the right sphere and has the lowest average F-score. OPTICS and DBScan remove all points where the density level is lower than the density level in the overlap. K-Means performs better than OPTICS and DBScan but, still, incorrectly assigns points of the right sphere to the left sphere. RIC correctly assigns border points of spheres in non-overlap areas but, fails to do that for many border points in the overlap. CORE separates clusters along the center of the overlap and achieves the highest average F-score. Varying the radius and density has similar results: CORE is more accurate in separation of the overlapping clusters.

3.9.2 Performance Evaluation

The thirty spheres dataset is five dimensional and consists of thirty non-overlapping, differently sized and randomly placed spheres. The last line in Table 3.2 shows that all methods perform well for this dataset. We use this dataset to empirically evaluate the performance and the dependence on the sample size for each method.

Figure 3.9.2 evaluates CORE for different sample sizes on the thirty spheres dataset which consists of 10^5 data points. The results present here are similar to results achieved with other datasets.

CORE outperforms other methods and produces better quality clusters for the same sample size (cf. Figure 3.6(a)). The quality of CORE monotonically decreases as the size of the sample becomes smaller. The quality of DataBubbles using OPTICS is the lowest because identification of valleys in reachability plot computed using data bubbles is not effective with steepness parameter of OPTICS. The computation time of CORE (cf. Figure 3.6(b)) is similar to CURE which uses B-Tree for fast computation of the neighborhood, and less than OPTICS or DBScan which are implemented with DataBubbles but without any indexing structure. Supporting their implementation with an indexing structure would bring them to the level of CORE.



(a) (b)

Figure 3.6: Comparison of Performance

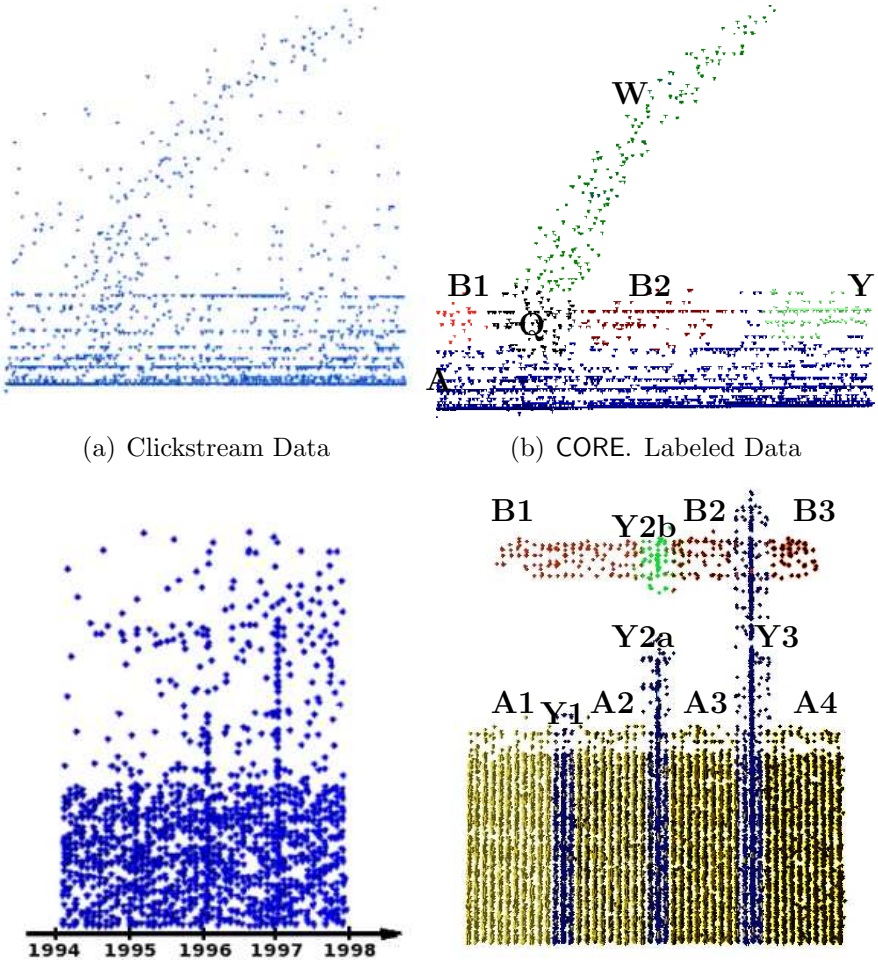
3.9.3 Real World Data

Figure 3.7 evaluates CORE for two real world datasets that contain time-varying data: the web log dataset and the financial dataset. Datasets of these type are usually very large and often requires precise quality clustering which is a challenge for parametric techniques.

Web logs. The dataset contains information about the web page accesses handled by the `sunsite.dk` server, a mirror of open source projects. The server is widely used in Europe and generates a clickstream log of more than 100MB per day from more than 250 countries. Figure 3.7(a) shows the clicks received from `.com` domain for one day. The X coordinate shows the time of the click and the Y coordinate shows the URL of the retrieved document. URL's are ordered according to their time of first occurrence. This yields curves for the click of search robots who scan entire collections of web documents. Search engines aim to scan the entire collection of web documents resulting in a curve, while humans tend to visit a few selected documents only resulting in the horizontal lines.

CORE successfully identifies activity of a search robot in the dataset (cf. cluster W in Figure 3.7(b)) and separates it from the main trend (cf. cluster A in Figure 3.7(b)). Other techniques produce less quality results: merge W with A, split A or remove many border points from both clusters.

Financial Data. We use a database with 150000 transactions by more than 4500 clients of a bank over a period of four years. Each point in Figure 3.7(c) indicates one transaction. The Y coordinate denotes the account identifier and the X coordinate corresponds to the day of the transaction. The data reveals two patterns in the behavior of clients: (i) the activity and the number of clients increases over the years (Figure 3.7(c) has more points to the right than to the left) and (ii) there are clients with many and clients with few transactions (points



(a) Clickstream Data (b) CORE. Labeled Data

(c) Financial Data (d) CORE. Labeled Data

Figure 3.7: Real World Datasets

are dense at the bottom of Figure 3.7(c)).

Besides the above patterns we can see intersection patterns at the beginning of each calendar year. At these times all customers show an increased activity. For a detailed analysis it is useful to separate the data and investigate the beginning-of-the-year activities independently from the rest of the data.

Comparing to other techniques CORE identifies all intersection patterns (cf. Figure 3.7(d)). Other techniques split cluster Y3 and its parts merge with clusters $B2$, $B3$, $A3$ and $A4$.

3.10 Conclusions and Future Work

This chapter presents CORE a new nonparametric clustering technique for large numeric databases. CORE explicitly computes maxima of the density and represents them with cores. The computation of cores is based on the AD-Tree, an incrementally constructed density estimation of the data. Key properties of the AD-Tree are a minimal adaptive grid that does not oversplit the space. CORE builds on this property and introduces rectangular neighborhoods, which, together with gradients, are used to reliably identify cores. The experimental results show that CORE outperforms other clustering methods–particularly for datasets with clusters that overlap or vary in density. In the future work we want to generalize the definition of cores to permit core with varying densities and explore CORE on datasets which are specific to certain areas (medical, spatial data).

Chapter 4

Separation of Overlapping Clusters

Current clustering techniques aim to find dense areas and return these areas as clusters of the data. This works well for non-overlapping clusters, but yields poor results if clusters overlap. In this chapter we describe how CORE can be extended to separate overlapping clusters.

CORE models clusters in terms of their cores: connected areas of locally maximal density. In the presence of overlapping clusters cores divide each other into *fragments* and *overlaps*. In order to reconstruct complete cores we compute the cores of overlapping clusters in two steps: first we use CORE to compute fragments and overlaps of cores and then we connect fragments that are separated by an overlap. In order to connect fragments we consider information about their density and direction. The experimental results show that our method successfully separates overlapping clusters. We analytically investigate the separation for the family of polynomial functions and identify the conditions under which clusters with polynomial density functions can be separated.

4.1 Introduction

Clustering techniques partition a set of tuples into non-overlapping groups. Each tuple represents a data point (or observation) in d-dimensional space. Typically, data points are partitioned according to the density of the data: clusters correspond to dense regions separated by low density regions. This approach works well for non-overlapping clusters, but yields poor results for clusters that overlap. In regions where clusters overlap the density is higher than in the surrounding regions and the overlap is identified as a cluster on its own. In order to separate overlapping clusters we need to consider all density levels of the data. Dense regions separated by higher density regions are *overlaps* and dense regions separated by lower density regions are *fragments* of clusters.

In this chapter we extend CORE to overlapping clusters. In the first step CORE determines the clusters on all levels of the density and represents them with cores.

In presence of overlapping clusters, the cores intersect and divide each other into fragments and overlaps. In the second step, CORE exploits the dimensionality and orientation of cores to reconstruct complete cores from fragments and overlaps.

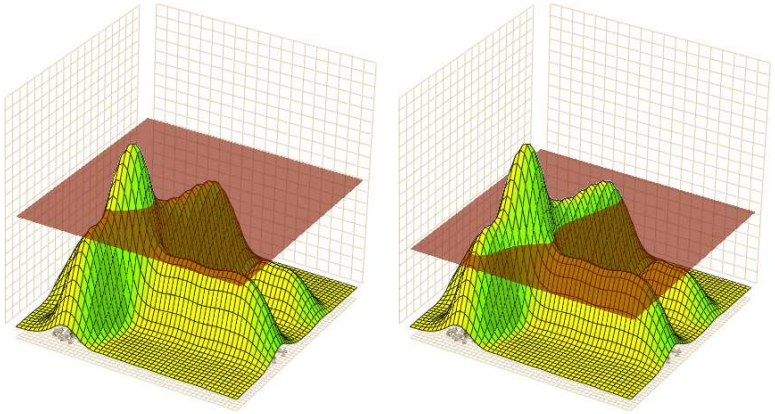
We distinguish two types of overlaps between clusters: first, clusters overlap but their cores do not intersect and, second, cores of clusters intersect. Clusters with the first type of overlap are not divided into fragments and the overlap area may or not produce new local maxima in the density. If no new local maxima is produced, then the major of clustering techniques find the original clusters with no overlaps and, possibly, incorrectly assign data point of the overlap to one of the clusters. If there is a new maxima, then all clustering techniques identify overlap area as a separate cluster or merge clusters. In our work we address the challenge of separating clusters with the second type of overlap. Overlaps of these type divide cores of clusters into fragments and always produce a new local maxima in the density. Separation of clusters with the first type of overlap is a complex issue, because clusters do not divide each other into parts. In general, for successful identification of first type overlaps the domain knowledge is required. In our work we analytically establish conditions for identification of first type overlaps between overlapping clusters with polynomial density function.

Overlapping clusters occur in many datasets and are particularly common in time-varying data. Consider the amount of money spent over time. Typically, each person spends a small amount each day resulting in an almost constant trend (cluster) over the whole year. In addition, people spend larger amounts at the end of the week and at the beginning of new seasons. This results in clusters that are perpendicular to and embedded in the clusters that model regular daily expenses. As another example consider the income of a person. Some people start to work and earn money immediately after high school, which yields a gradual increase of the income over time. Other people start to work after the university, which yields a late but more pronounced increase of the income. This results in line-shaped clusters that intersect at different angles.

The chapter makes the following contributions.

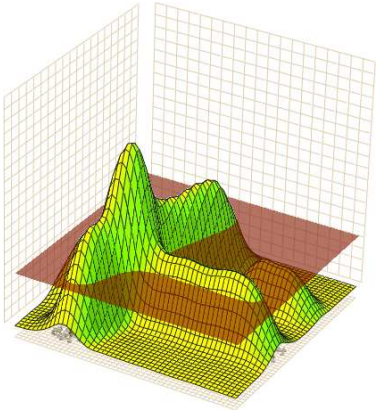
- We classify cores as *fragments*, *overlaps* and *complete cores*, respectively.
- We define how to combine fragments and overlaps into complete cores, and label database points by assigning them to complete cores based on gradient paths.
- We analytically investigate the separation of clusters for the family of polynomial functions and identify the conditions under which clusters with polynomial density functions can be separated.
- We experimentally evaluate our method for synthetic and real world data, and show that it is invariant to the dimensionality, angle of intersection, curvature, and size of clusters.

The chapter proceeds as follows. Section 4.2 defines the problem of separating overlapping clusters and introduces notation used in the Chapter. Section 4.3 discusses related work. Section 4.4 introduces the notion of overlapping clusters and defines key concepts to their separation: fragments and overlaps of cores. Section 4.5 explains reconstruction of cores from fragments and overlaps and Section 4.6 defines labeling of data points. Analytical investigation is given in Section 4.8. We give a detailed experimental evaluation for synthetic and real world data in Section 4.9. Finally, Section 4.10 concludes the chapter and offers future work.



(a) Overlap is Identified as Only Cluster in the Data.

(b) Overlap and One of the Overlapping Cluster are Merged.



(c) Two Overlapping Clusters are Merged

Figure 4.1: The Challenge of Separation of Overlapping Clusters. The Figures Illustrate the Output of DBScan Clustering.

4.2 Problem Definition

In order to illustrate the problem of separating overlapping clusters consider Figure 4.1, which shows the density function of a two dimensional dataset with three line-shaped clusters. Two of the clusters overlap and each of these clusters is divided into three parts: two fragments and an overlap. The density is highest in the overlap area where the densities of both clusters accumulate. Common clustering techniques find clusters by identifying maximally connected areas at an application-specified density level. Such an approach allows three possible clusterings. Figure 4.1(a) illustrates the clustering if a high density level is chosen (the horizontal plane represents the density level at which we cut the density function of the data). In this case only the overlap area is identified as a cluster. Figure 4.1(b) illustrates a medium density level. In this case the overlap is merged with one of the overlapping clusters. Figure 4.1(c) illustrates a small density level. In this case the overlapping clusters are identified as a single cluster. Note that no single density level exist at which the overlapping clusters are separated.

In summary, with overlapping clusters we cannot use clustering techniques that determine clusters at one density level only. Instead we must cluster the data at all density levels to get fragments and overlaps of clusters. The density level and orientation of fragments and overlaps is used to reconstruct complete clusters.

Problem definition: Let $D = \{t_1, t_2, \ldots, t_n\} \subset R^d$ be a database with n d-dimensional tuples. Assume D consists of k clusters that may overlap. Our goal is to identify and separate these k clusters.

Consider the three clusters in Figure 4.2. The data points of each cluster are spread around its core. In the example, data points of cluster C_1 are distributed around curve E_1. E_1 is the core of the cluster since it is the local maxima of the density function. The cores of the other two clusters intersect and divide each other into *fragments* and *overlaps*. In the example, the cores of clusters C_2 and C_3 divide each other into fragments F_{21} and F_{22} (belonging to the core of C_2), F_{31} and F_{32} (belonging to the core of C_3), and overlap O (belonging to cores of C_2 and C_3). Note that the density of the overlap area is the sum of the densities of the intersecting clusters and fragments of the same core have the same orientation near the overlap area. These are the key properties that we exploit to reconstruct complete cores from fragments and overlaps.

We use C to denote clusters in the dataset, E to denote the cores of clusters, F to denote fragments of cores, and O to denote intersections of cores.

4.3 Related Work

We discuss related work in three areas of clustering: density based clustering, subspace clustering, and hierarchical clustering. For all techniques we focus on

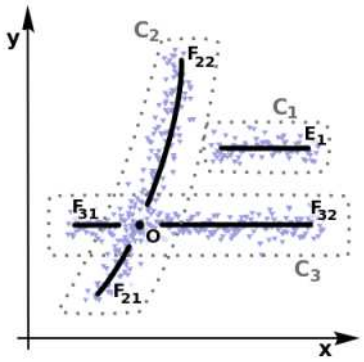


Figure 4.2: Cores, Fragments and Overlap

their potential to separate overlapping clusters.

DBScan [17] is a classical density based clustering technique. It groups database points that are connected to each other with a density level that exceeds a certain threshold. DBScan finds clusters of arbitrary shape, effectively identifies outliers, and is robust to noise. Other density based techniques improve DBScan in terms of the input parameters. DBCLASD [65] aims to estimate the best optimal parameters, while OPTICS [3] enables efficient computation of clusters for broad range of input parameters. Density based clustering has also been extended with signal processing techniques [54] and kernel based techniques [28]. [55] identifies a cluster with a core, which is the most dense area of the cluster. [17], [65], [54] and [28] find clusters only at one density level and, hence, cannot separate overlapping clusters. OPTICS [3] computes clusters at all density levels, however, does that by projecting multidimensional data into one dimensional space. Therefore, OPTICS is not able to separate overlapping clusters, since, it is not possible to restore orientation of clusters from their one-dimensional projection.

Subspace clustering [2, 34, 63, 41, 68] assumes that the dimensionality of clusters is lower than the dimensionality of the dataset. The techniques aim to find all subspaces that contain clusters. However, in low-dimensional spaces the orientation of clusters is lost and non-overlapping fragments of clusters overlap. CURLER [59] improves the situation and finds nonlinear correlation clusters. Together with a cluster, CURLER returns the smallest subspace that the orientation of the cluster is present. CURLER cannot separate overlapping clusters because of following reasons: first, it finds clusters only at one density level and, hence, either merges overlapping clusters or only finds their overlap and, second, it computes global orientation of clusters. Overlapping clusters can have the same global orientation and, hence, we need to consider local orientation of fragments near the overlap.

Hierarchical clustering organizes the data points into a hierarchy starting with single point clusters at the leaves and one cluster at the root. Hierarchical clustering techniques for very large databases [69] and arbitrary shaped clusters [24]

have been proposed. Hierarchical clustering allows users to find different numbers of clusters without the re-computation of the model. The hierarchical clustering techniques fails to separate overlapping clusters because (*i*) they do not consider orientation of clusters and (*ii*) at any level of the hierarchy fragments are not separated from the overlaps.

4.4 Overlapping Clusters

This section formalizes cores, fragments and overlaps. In Section 3.5 we give procedural definition of cores. We define cores with a connected set of grid points in the AD-Tree which best approximate local maxima. In this section we give declarative definition of cores that is independent of the AD-Tree.

We define cores with a help of core points:

Definition 4.1. *[Core points]. Let R^d be the domain of the density function f of the data. $e \in R^d$ is a core point if the following conditions are satisfied.*

(*i*) $f(e) \geq f(x)$ *for some* $\varepsilon > 0$ *and for all* $x \in S_\varepsilon(e)$, *where* $S_\varepsilon(e) = \{x \in R^d : \|e - x\| < \varepsilon\}$ *is the* ε *neighborhood of* e.

(*ii*) $f(e) > 0$

Condition (i) ensures that e is a local maxima: the density in the neighborhood is less than or equal to $f(e)$. The density is strictly lower for non-core points and is the same for core points of the neighborhood. Condition (ii) filters out noise.

Example 4.1. *[Core points]. Consider a one dimensional dataset with the density function illustrated in Figure 4.3. There are five cores in the example: line-*

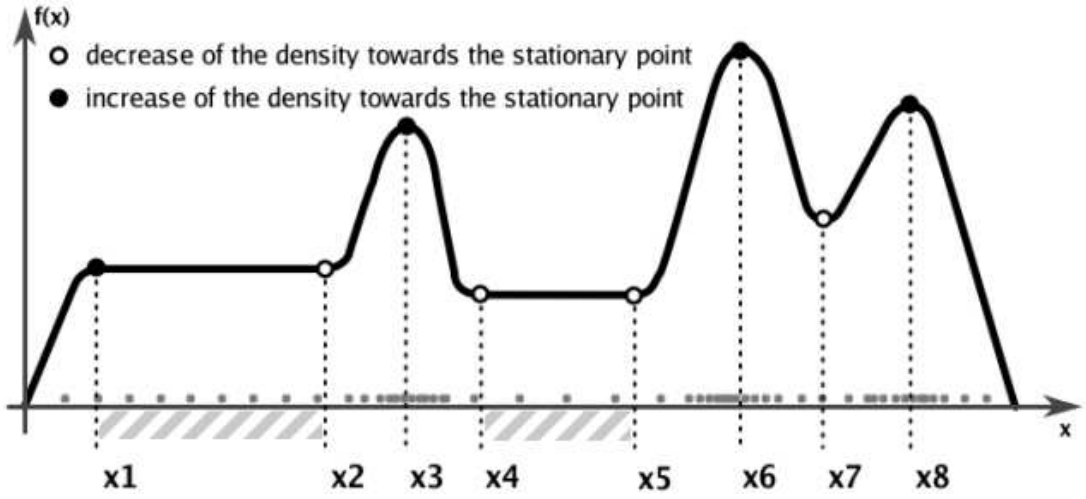
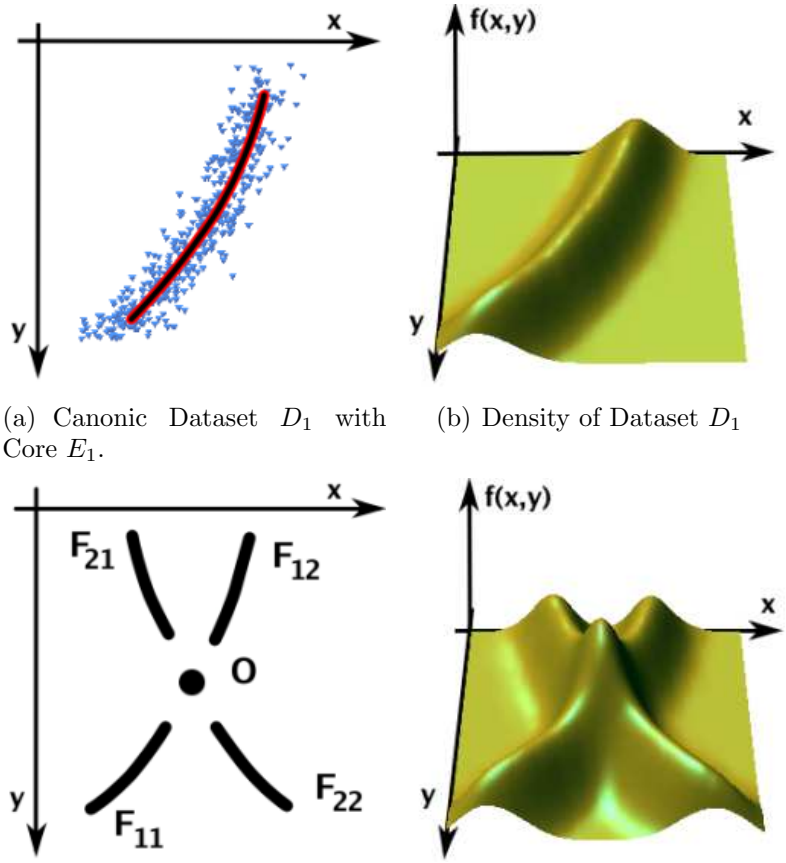


Figure 4.3: Cores in One-dimensional Data

cores: $[x_1, x_2)$, (x_4, x_5) *and point-cores:* x_3, x_6, x_8. *Note that* x_2 *is not a core point. This is because* $f(x_2) < f(x)$ *for all* $\varepsilon > 0$ *and some* $x \in S_\varepsilon(x_2)$.

Example 4.1 illustrates the relationship between extrema points of the density function and core points. Point x *is an extrema point iff* $f'(x) = 0$. *Core points are extrema points, but not all extrema points are core points. For example,* x_7 *is an extrema point (local minima), however, it is not a core point (condition (i) is not satisfied). Similarly, points* x_4 *and* x_5 *are extrema points but not core points.*



(a) Canonic Dataset D_1 with Core E_1.

(b) Density of Dataset D_1

(c) Fragments and Overlaps in Dataset D. $D = D_1 \cup D_2$.

(d) Density of Dataset D.

Figure 4.4: Fragments and Overlaps of Cores.

Definition 4.2. *[Core.] Let* D *be a dataset with one connected local density maxima and* $E \subset R^d$ *be a non-empty set.* E *is a core iff*

$$x \text{ is a core point} \iff x \in E.$$

Definition 4.3. *[Fragments of cores]. Let* D_1 *and* D_2 *be datasets with one connected local density maxima each. Let* E_1 *be the core of database* D_1 *and* E_2 *be the core of database* D_2. *Let* $D = D_1 \cup D_2$. $F \subset E_1 \cup E_2$ *is a fragment iff*

(i) F *is a connected set*

(ii) $F \cap (E_1 \cap E_2) = \emptyset$

(iii) F *is maximal, i.e., for all* $F' \supset F$ *either (i) or (ii) is not satisfied.*

Definition 4.4. *[Overlap of cores]. Let* D_1 *and* D_2 *be datasets with one connected local density maxima each. Let* E_1 *be the core of database* D_1 *and* E_2 *be the core of database* D_2*. Let* $D = D_1 \cup D_2$*.* O *is an overlap iff*

(i) $O \subset E_1 \cap E_2$

(i) O *is a connected set*

(iii) O *is maximal, i.e., for all* $O' \supset O$ *either (i) or (ii) is not satisfied.*

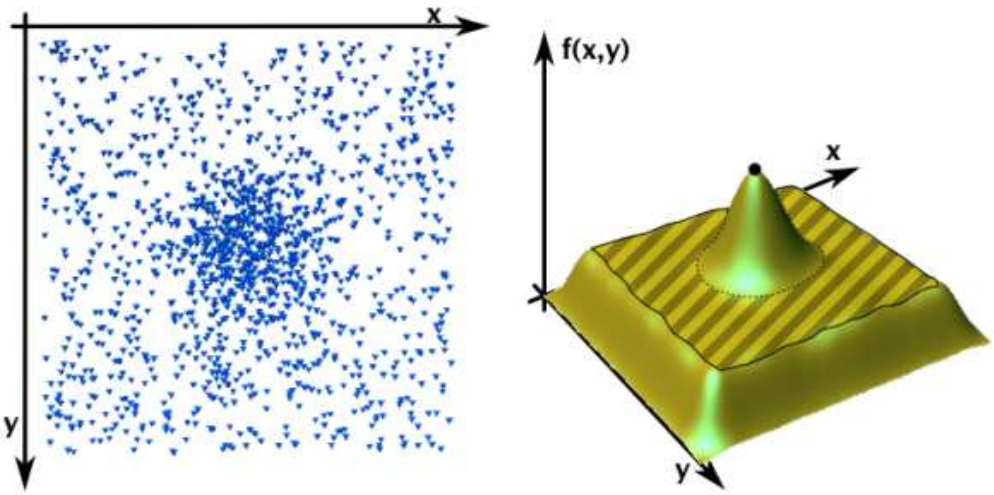
As an example of fragments and overlaps consider Figure 4.4. Figure 4.4(a) illustrates a two-dimensional dataset D_1 with one connected local density maxima. The data points of the dataset form one cluster and are distributed around a line. Line E_1 is the core of the cluster since the density is locally maximal at E_1. Figure 4.4(b) shows the density function of D_1. Figure 4.4(c) illustrates dataset D, which is a union of datasets D_1 and D_2 with line-cores E_1 and E_2. The cores divide each other into fragments and overlaps: core E_1 divides E_2 into F_{21} and F_{22}; core E_2 divides E_1 into fragments F_{11} and F_{12}; cores E_1 and E_2 have overlap O in common. Note, that the density is the same for fragments of the same core, and the density of the overlap is the sum of the densities of the intersecting cores.

Core E is d-dimensional if E is a d-dimensional set. We use the topological approach to define the dimensionality of sets. Intuitively, set A is d-dimensional if it is a bending of the R^d space. For example, A is an elementary d-dimensional set, if it is a bending of R^d. A set is a (general) d-dimensional set if it is a result of a number of bendings of R^d.

Definition 4.5. *[Dimensionality of Sets]. Let* $A \subset R^d$*.* A *is an elementary set of dimensionality* d *if there exists a homeomorphic (one-to-one, continuous inverse) function from* A *to* R^d*.* A *is a set of dimensionality* d *iff* A *is a connected set of points and any neighborhood of* A *is an elementary set of dimensionality* d*.*

Example 4.2. *[Dimensionality of Cores]. Consider dataset* D *in Figure 4.4(c) and the corresponding density function in Figure 4.4(d). The core that consists of fragments* F_{11}*,* F_{12} *and overlap* O *is a one dimensional core.*

Note, that cores of different dimensionality can overlap or enclose other cores. Figure 4.5(a) illustrates this for a dataset with two clusters: a square (two-dimensional core) and a sphere located in the middle of the square (one-dimensional core). Figure 4.5(b) shows the density function of the dataset. There are two local maxima of the density function and therefore two cores in the data: the center of the sphere is a one-dimensional core (center point in Figure 4.5(b)), and the square area with a hole in the middle is a part of a two-dimensional core (the striped area in Figure 4.5(b)) that encloses a one-dimensional core.



(a) Scatter Plot (b) Density Function and Cores

Figure 4.5: Cores in Two-Dimensional Data

4.5 Reconstruction of Complete Cores

The concepts of cores, fragments, and overlaps are based on a dataset $D = D_1 \cup D_2 \cup \cdots \cup D_k$, where each D_i contains exactly one core. In practice only D is given and the D_i are not known. In order to compute complete cores we need to determine local density maxima, classify them into fragments and overlaps, and then restore complete cores from fragments and overlaps.

For the computation of local density maxima we use CORE clustering. CORE computes local density maxima at all levels of the density and explicitly represents them with a connected set of grid points $\mathsf{X}_i = \{\lambda_{J_1}^{k_1}, \lambda_{J_2}^{k_2}, \dots\}$ in the AD-Tree. In the case of non-overlapping clusters, each local density maxima X_i approximates the complete core E_i of cluster C_i. In case of overlapping clusters, each X_i is an approximation of a fragment or overlap of one of the overlapping cores in the dataset. We denote approximated fragments by $\hat{F}_i$ and assume that it corresponds to a density peak X_i computed by CORE.

Our approach restores complete cores based on the orientation of cores. Intuitively, we need to consider all possible triplets of cores $(\mathsf{X}_1, \mathsf{X}_2, \mathsf{X}_3)$, and connect cores X_1 and X_3 into a complete core if *i*) core X_2 is an overlap between X_1 and X_3, and *ii*) cores X_1 and X_3 have similar orientation at the border with core X_2. We define the border between fragments and their orientation with a help of the gradient path.

Figure 4.6 illustrates the gradient path (cf. Definition 3.10) computed in a two dimensional AD-Tree at grid point $\lambda_{3,1}^3$. The gradient path is a sequence of grid points $P(\lambda_{3,1}^3) = (\lambda_{3,1}^3, \lambda_{1,2}^2, \lambda_{1,1}^2, \lambda_{2,1}^2, \lambda_{3,1}^2)$ such that the gradient at each grid point is pointing towards the next grid point of the gradient path. For example, in Figure 4.6, the solid arrows denote the gradients at the grid points of the

gradient path from $\lambda^3_{3,1}$ to $\lambda^2_{3,1}$. According to Definition 3.10 we use the following strategy to incrementally compute the next grid point on a gradient path. We take the last grid point λ^k_J of the gradient path and compare the angles between gradient $\nabla\hat{f}(\lambda^k_J)$ at λ^k_J and the directions from λ^k_J to its neighboring grid points. The neighboring grid point that yields the smallest angle with the gradient is the next point of the gradient path.

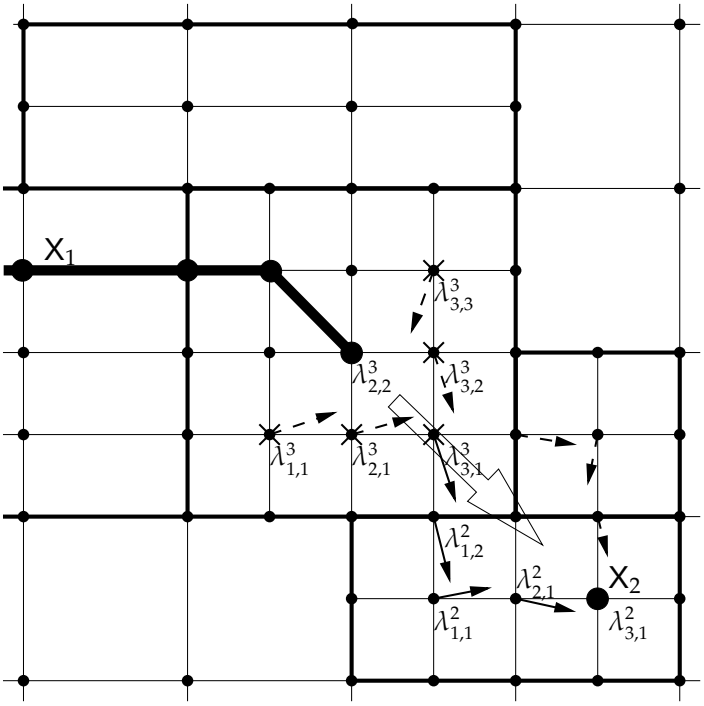


Figure 4.6: Illustration of Gradient Path, Border, and Orientation.

Core X_1 has a border with core X_2 if X_1 is connected to X_2 by a gradient path, i.e., if there exists a grid point in the neighborhood of core X_1 where a gradient path starts that ends at a grid point of core X_2. The set of all grid points in the neighborhood of core X_1 where a gradient path starts that ends at a grid point of core X_2, is the border of core X_1 with core X_2.

Definition 4.6. *[Border of a Core]. Let* X_1 *and* X_2 *be two cores. A set of grid points,* $N(\mathsf{X}_1 \rightarrow \mathsf{X}_2)$*, is a border of core* X_1 *with core* X_2 *iff*

$$N(\mathsf{X}_1 \rightarrow \mathsf{X}_2) = \{\lambda^k_J \in \mathsf{R}(\mathsf{X}_1), P(\lambda^k_J) \in \mathsf{X}_2\},$$

where $\mathsf{R}(\mathsf{X}_1)$ *are grid points which are in the rectangular neighborhood with grid points of core* X_1*, i.e.:*

$$\mathsf{R}(\mathsf{X}_1) = \{\lambda^{k_1}_{J_1} : \lambda^{k_1}_{J_1} \in \mathsf{R}(\lambda^{k_2}_{J_2}), \lambda^{k_2}_{J_2} \in \mathsf{X}_1\}$$

, and $P(\lambda^k_J)$ *is a gradient path at grid point* λ^k_J*.*

There is no border between cores X_1 *and* X_2 *iff* $N(\mathsf{X}_1 \rightarrow \mathsf{X}_2) = \emptyset$*.*

As an example of a core with a border consider Figure 4.6. There are five grid points $\lambda^3_{1,1}$, $\lambda^3_{2,1}$, $\lambda^3_{3,1}$, $\lambda^3_{3,2}$ and $\lambda^3_{3,3}$ in the neighborhood of core X_1 which have gradient paths pointing to grid point $\lambda^2_{3,1}$ of core X_2. Thus, the border of core X_1 with core X_2 is $N(\mathsf{X}_1 \rightarrow \mathsf{X}_2) = \{\lambda^3_{1,1}, \lambda^3_{2,1}, \lambda^3_{3,1}, \lambda^3_{3,2}, \lambda^3_{3,3}\}$. Note that the border is directed. If core X_1 is connected to core X_2 by a gradient path and, hence, has a border with this core, then core X_2 is not connected to X_1 by a gradient path and, thus, does not have a border with this core. For example, in Figure 4.6 core X_2 does not have a border with core X_1.

The orientation of core X_1 at the border with core X_2 is the average of all gradients on gradients paths that connect X_1 with X_2.

Definition 4.7. *[Orientation of Cores X_1 at the Border with Core X_2].*

Let X_1 and X_2 be two cores such that there is a border between X_1 and X_2, i.e., $N(\mathsf{X}_1 \rightarrow \mathsf{X}_2) \neq \emptyset$. Let $\Omega(\mathsf{X}_1 \rightarrow \mathsf{X}_2)$ be a set of all grid points that belong to gradient paths connecting core X_1 with X_2:

$$\Omega(\mathsf{X}_1 \rightarrow \mathsf{X}_2) = \{\lambda^k_J : \lambda^k_J \in P(\lambda^{k'}_{J'}) \text{ and } \lambda^{k'}_{J'} \in N(\mathsf{X}_1 \rightarrow \mathsf{X}_2)\}$$

The orientation of core X_1 at the border with X_2 is a d-dimensional vector

$$\nabla \hat{f}(N(\mathsf{X}_1 \rightarrow \mathsf{X}_2)) = \frac{1}{|\Omega(\mathsf{X}_1 \rightarrow \mathsf{X}_2)|} \sum_{\lambda^k_J \in \Omega(\mathsf{X}_1 \rightarrow \mathsf{X}_2)} \nabla \hat{f}(\lambda^k_J)$$

In Figure 4.6 the white arrow denotes the orientation $\nabla \hat{f}(N(\mathsf{X}_1 \rightarrow \mathsf{X}_2))$ of core X_1 at the border with X_2. The orientation is the average of all gradients in Figure 4.6 and represents the direction in which core X_1 connects to core X_2.

Definition 4.8. *[Reconstruction of Complete Cores]. Let X_1, X_2 and X_3 be cores. Then, cores X_1, X_2 and X_3 belong to the same complete core $\hat{E}$, and X_2 is an overlap between cores X_1 and X_3 if the following conditions are satisfied:*

1. *Cores X_1 and X_3 have a non-empty border with core X_2, i.e.,*

$$N(\mathsf{X}_1 \rightarrow \mathsf{X}_2) \neq \emptyset \text{ and } N(\mathsf{X}_3 \rightarrow \mathsf{X}_2) \neq \emptyset$$

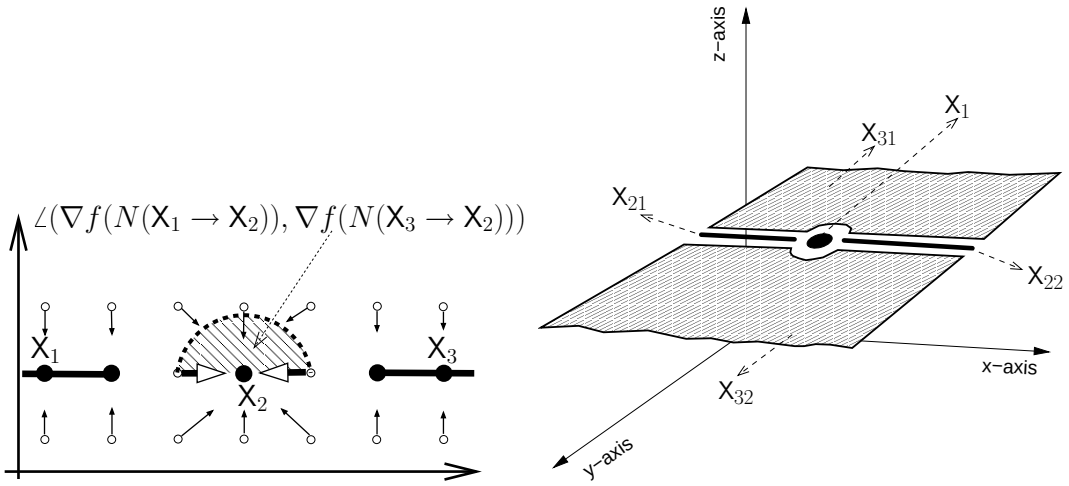
2. *The densities of cores X_1 and X_3 are roughly the same:*

$$|\hat{f}(\mathsf{X}_1) - \hat{f}(\mathsf{X}_3)| < \epsilon$$

3. *Orientation vectors of cores X_1 and X_3 at the border with X_2 are pointing towards each other:*

$$\angle\big(\nabla f(N(\mathsf{X}_1 \rightarrow \mathsf{X}_2)), \nabla f(N(\mathsf{X}_3 \rightarrow \mathsf{X}_2)\big) > 90^\circ$$

Definition 4.8 requires three conditions to be fulfilled for a triplet of cores that belong to the same complete core. The first condition requires that two cores X_1 and X_3 from a given triplet have a border with another core X_2. Thus, we require that core X_2 is an overlap between cores X_1 and X_3. The second condition requires, that cores X_1 and X_3 have roughly the same density level. The last condition of Definition 4.8 requires that cores X_1 and X_3 are similarly oriented at the border with the overlap, i.e., their orientation vectors are pointing towards each other.



(a) Third Condition of Definition 4.8.

(b) Overlapping Cores of Different Dimensionality.

Figure 4.7: Reconstruction of Complete Cores.

Figure 4.7(a) illustrates Definition 4.8. In Figure 4.7(a) X_2 is an overlap which divide complete core into cores X_1 and X_3. The angle between the orientation vectors of cores X_1 and X_3 is 180° and, hence, the third condition of Definition 4.8 is satisfied. Figure 4.7(b) illustrates the reconstruction of complete cores with different dimensionality. There are five cores in the figure. Core X_1 (0-dimensional core) is the overlap which divides a one-dimensional complete core into cores X_{21} and X_{22}. Cores X_{21}, X_{22} and X_1 are overlaps which divide a two-dimensional complete core into cores X_{31}, X_{32}. Therefore, the reconstructed cores are $\hat{E}_2 = \mathsf{X}_{21} \cup \mathsf{X}_{22} \cup \mathsf{X}_1$ and $\hat{E}_3 = \mathsf{X}_{31} \cup \mathsf{X}_{32} \cup \mathsf{X}_{21} \cup \mathsf{X}_{22} \cup \mathsf{X}_1$.

4.6 Labeling the Data

This section describes the labeling of data points. We scan the dataset $D = \{t_1, t_2, \ldots, t_n\}$ and use complete cores and overlaps to label each data point with a cluster ID. The key idea for labeling data point t_s is the following. Let $t_s \in D$ be a dataset point and λ be the closest grid point to t_s. If the gradient path $P(\lambda)$ points to complete core $\hat{E}$ then t_s is labeled with $\hat{E}$. If the gradient path $P(\lambda)$

points to overlap X_o then we label t_s with one of the cores that intersects X_o using a randomized model.

Definition 4.9. *[Labeling of t that has a path to complete core $\hat{E}$]. Let $t_s \in D$ be a point and $\hat{E}$ be a complete core of D. t_s is labeled with $\hat{E}$ iff*

(i) $\lambda \in$ *AD-Tree such that* $\|\lambda - t_s\| = \min_{\lambda_J^k \in \textit{AD-Tree}} \|\lambda_J^k - t_s\|$

(ii) $P(\lambda) \in \hat{E}$.

Definition 4.10. *[Labeling of t_s that has a path to overlap X_o]. Let X_o be an overlap for a set of complete cores $\{\hat{E}_1, \hat{E}_2, \ldots, \hat{E}_l\}$, i.e.,*

$$f(\mathsf{X}_o) \approx \sum_{i=1}^{l} f(\hat{E}_i).$$

Let $t_s \in D$ be point with a gradient path to overlap X_o:

(i) $\lambda \in$ *AD-Tree such that* $\|\lambda - t_s\| = \min_{\lambda_J^k \in \textit{AD-Tree}} \|\lambda_J^k - t_s\|$

(ii) $P(\lambda) \in \mathsf{X}_o$.

Then t_s is labeled with one of the complete cores $\hat{E}_i$ in the following way. Let Z be a random generator that generates values $\{1, 2, \ldots, l\}$ with the following probabilities: $f_Z(i) = f(\hat{E}_i)/f(\mathsf{X}_o)$, $i \in \{1, 2, \ldots, l\}$. We label t_s with $\hat{E}_i$ if the generator Z outputs value i.

Example 4.3. *[Labeling of the Data].*

Consider the two dimensional dataset in Figure 4.8. There are two intersecting

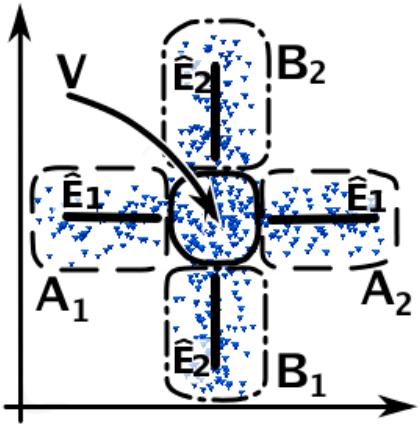


Figure 4.8: Labeling the Data

complete cores: $\hat{E}_1$ and $\hat{E}_2$. The paths of the tuples in A_1 and A_2 point to $\hat{E}_1$,

therefore the points are labeled with $\hat{E}_1$. Similarly the paths of the tuples in B_1 and B_2 point to $\hat{E}_2$ and therefore they are labeled with $\hat{E}_2$. The paths of the tuples in V point to the overlap area. Since the density at both $\hat{E}_1$ and $\hat{E}_2$ cores is equal, half of the points of V are labeled with $\hat{E}_1$ and half of the points of V are labeled with $\hat{E}_2$.

4.7 Algorithms

Algorithm 8 implements the reconstruction of complete cores. As input the algorithm takes a set of cores $\mathsf{X} = \{\mathsf{X}_1, \mathsf{X}_2, \dots, \}$ computed by the *computeCore* (cf. Algorithms 6), the AD-Tree of the dataset and precision threshold ϵ. The algorithm reconstructs complete cores in three steps. First (cf. lines 2–8), the algorithm classifies cores into fragments and overlaps. The algorithm traverse through border grid points of each core X_i and computes the gradient path. If the gradient path points to another core X_o, then the algorithm classifies X_o as an overlap and X_i as a fragment of some core separated the overlap X_o. To avoid multiple computation, for each overlap X_o the algorithm stores a set of associated fragments $\hat{F}(O)$ of cores separated by overlap X_o, and grid points $\Omega(\mathsf{X}_i \to \mathsf{X}_o)$ of gradients paths which connect fragment X_i with overlap X_o. Next (cf. lines 9–20), the algorithm iterates through identified overlaps and restores complete cores from the associated cores. Lines 10–12 compute orientation vectors of each associated core. Lines 13–18 merge fragments into one core if they satisfy conditions of Definition 4.8. Line 19 declassifies core X_o from being an overlap if no complete cores are restored from the set of associated fragments $\hat{F}(\mathsf{X}_o)$. At last (cf. line 21), all cores, which are not yet a part of any complete core, the algorithm marks as individual complete cores in the dataset.

Figure 9 presents the algorithm to label the dataset points. The input of the algorithm are dataset D, the AD-Tree computed for D, and sets of cores $\hat{E}$ and overlaps $\hat{O}$ computed by Algorithm 8. We scan the dataset. For each data point we find the closest grid point λ_J^k in the AD-Tree and check whether gradient path $P(\lambda_J^k)$ points to an overlap or to a core. If $P(\lambda_J^k)$ points to an overlap, we assign the data point according to the random model to the intersecting cores. Otherwise, we label the point with $P(\lambda_J^k)$.

4.8 Analytical Evaluation

This section analytically investigates the separation of overlapping clusters. We show that we find all core points correctly in non-overlap areas of clusters (no false negatives). In overlap areas we identify all core points, but in addition we might also find points that are not cores (false positives).

Theorem 4.1. *[No false negatives.] Let Δ be the maximal distance between two*

Algorithm 8: restoreCores($\mathsf{X} = \{\mathsf{X}_1, \mathsf{X}_2, \dots\}$, AD-Tree, ϵ)

```
   /* initialize complete cores and overlaps                          */
 1 Ê = ∅ and Ô = ∅;

   /* classify cores into overlaps and fragments                      */
 2 for λ^k_J, X_i : λ^k_J ∈ R(X_i) and X_i ∈ X do
 3     if P(λ^k_J) ∈ X_o : X_o ≠ X_i then
           /* mark X_o as an overlap                                  */
 4         Ô = Ô ∪ X_o;
           /* mark X_i as a fragment and associate with X_o           */
 5         F̂(X_o) = F̂(X_o) ∪ X_i;
           /* update grid points of gradient paths which connect
              fragment X_i with overlap X_o                           */
 6         Ω(X_i → X_o) = Ω(X_i → X_o) ∪ λ^k_J;
 7     end
 8 end

   /* iterate through overlaps and restore complete cores             */
 9 for X_o ∈ Ô do
       /* compute orientation of the associated fragments             */
10     for X_i ∈ F̂(X_o) do
11         ∇f̂(N(X_i → X_o)) = 1/|Ω(X_i→X_o)| Σ_{λ^k_J ∈ Ω(X_i→X_o)} ∇f̂(λ^k_J);
12     end
       /* restore complete cores from the associated fragments        */
13     for X_1, X_2 ∈ F̂(X_o) do
14         if |f̂(X_1) − f̂(X_2)| < ε and
           ∠(∇f(N(X_1 → X_o)), ∇f(N(X_2 → X_o))) > 90° then
15             if ∃Ê_i ∈ Ê : X_1 ∈ Ê_i or X_2 ∈ Ê_i then Ê_i = Ê_i ∪ X_1 ∪ X_2;
16             else Ê = Ê ∪ {X_1, X_2};
17         end
18     end
       /* if no complete cores are restored remove the overlap        */
19     if X_m ∉ Ê_l for all X_m ∈ F̂(X_o) and Ê_l ∈ Ê then Ô = Ô/X_o;
20 end

   /* mark unassigned core as a complete core                         */
21 for X_i ∈ X : X_i ∉ Ê_l, Ê_l ∈ Ê do Ê = Ê ∪ {X_i};

22 return Ê and Ô;
```

Algorithm 9: labelData(D, AD-Tree, $\hat{E}$, $\hat{O}$)

```
1 for t_s ∈ D do
2     Find the nearest grid point λ_J^k ∈ AD-Tree to t_s;
      /* if gradient path point to an overlap then redistribute
         the points.  Otherwise, label data point with the core.
         */
3     if P(λ_J^k) ∈ Ô_i then
4         Label t_s according to Definition 4.10
5     else
6         Label t_s with core Ê_i : P(λ_J^k) ∈ Ê_i.
7     end
8 end
```

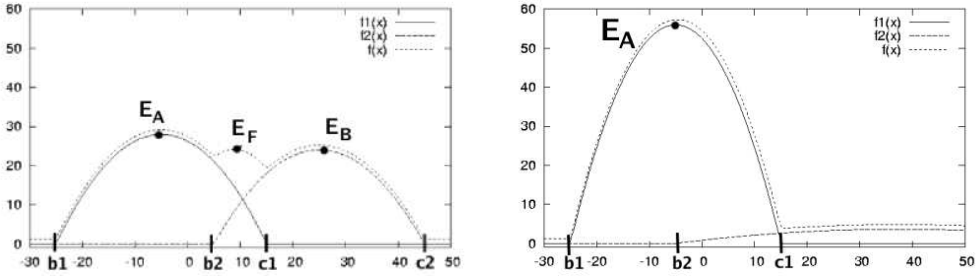
adjacent grid points in the AD-Tree. Let F be a fragment of a core, and CORE be an approximation of F computed by the CORE. Let $x \in F$ be a core point and $\hat{\alpha_x} \in$ CORE be the closest approximated core point to x: $\|\hat{\alpha_x} - x\| = \min_{\hat{\alpha}} \|\hat{\alpha} - x\|$. Then $\alpha_x \to x$ as $\Delta \to 0$.

PROOF: The proof derives from the definition of the derivative. □

Theorem 4.1 states that our approximation finds all core points as the granularity of grids in the AD-Tree increases.

Our method might find additional core points. This happens, since we model a database as the union of individual clusters and aim to restore the individual clusters. If two clusters are located close to each other and the density distribution of the clusters decreases slowly, the algorithm will find three cores with one false positive in the overlapping area.

Figure 4.9(a) illustrates the problem for a one dimensional dataset. The original database consists of two clusters with cores E_A and E_B. The density of the clusters decreases very slowly and therefore the overlap area of the clusters forms a false core E_F.



(a) False Positive (b) Different a_1 and a_2

Figure 4.9: Intersecting Parabolas

Proposition 4.8.1. [False positives] Let C_A and C_B be clusters, I be the overlap area of the clusters, E_A and E_B be the cores of the clusters and f_A and f_B be the density function of the clusters. Then the following observations hold:

- The probability of false positives decreases as the distance between cores E_A and E_B increases.
- The probability of false positives decreases as the speed (or degree, if f_A and f_B are polynomials) increases.
- The probability of false positives decreases as the difference between the steepness levels of f_A and f_B in I increases.

Below we give precise mathematical formulations of Proposition 4.8.1 for specific datasets and density functions. Section 4.8.1 investigates intersecting clusters with 0-dimensional cores. We examine two intersecting parabolas of degree l (the density function is of the form $f(x) = ax^{2l}$). We start the examination with one dimensional parabolas and next, we generalize the result to high dimensional parabolas. Section 4.8.2 discusses high-dimensional cores.

4.8.1 Zero-Dimensional Cores

This section formulates Proposition 4.8.1 for overlapping clusters C_A and C_B each having a 0-dimensional core and density function of the form:

$$f(x) = a_l x^l + a_{l-1} x^{l-1} + \cdots + a_1 x + a_0$$

As we will see below the degree of the polynomial has a substantial impact on the existence of false positives in the overlapping area. In order to isolate the impact of the degree of the polynomial and in order to simplify the mathematical expressions we investigate density function of the following form:

$$f(x) = ax^{2l}, \tag{4.1}$$

where $a < 0$ controls the steepness of the density function, and l is the degree of the polynomial. We assume that there are no cores in the overlap area I. We call f an lD parabola.

One-Dimensional Parabolas

The following investigates overlapping clusters with density functions of the form of Equation (4.1) and x being a $1D$ point.

Theorem 4.2. *[False positives for 1D parabolas.] Let C_A and C_B be two overlapping clusters. Let $f_1(x)$ and $f_2(x)$ be density functions of the clusters:*

$$f_1(x) = \begin{cases} c_1 + a_1(x-b_1)^{2l} \text{ if } -(\frac{c_1}{a_1})^{\frac{1}{2l}} < x - b_1 < (\frac{c_1}{a_1})^{\frac{1}{2l}} \\ 0 \text{ otherwise} \end{cases},$$

$$f_2(x) = \begin{cases} c_2 + a_2(x-b_2)^{2l} \text{ if } -(\frac{c_2}{a_2})^{\frac{1}{2l}} < x - b_2 < (\frac{c_2}{a_2})^{\frac{1}{2l}} \\ 0 \text{ otherwise} \end{cases},$$

I is the overlap area of the parabolas:

$$I = \Big[-\Big(\frac{c_2}{a_2}\Big)^{\frac{1}{2l}} + b_2, \Big(\frac{c_1}{a_1}\Big)^{\frac{1}{2l}} + b_1 \Big]$$

The overlap area I does not contain false positives iff

$$|I| < \Big(\frac{2l+1}{4}\Big)^{\frac{1}{2l+1}} \Big(\frac{{a_i}^{\frac{2}{4l^2-1}} - {a_j}^{\frac{2}{4l^2-1}}}{{a_j}^{\frac{1}{2l-1}}} \Big), \tag{4.2}$$

where $i, j \in \{1,2\} : i \neq j, a_j \geq a_i$, and $|I| = \Big(\frac{c_1}{a_1}\Big)^{\frac{1}{2l}} + b_1 + \Big(\frac{c_2}{a_2}\Big)^{\frac{1}{2l}} - b_2$. is the length of the overlap area.

PROOF: False positive E_F exists in $[c_1, b_2]$ iff

$$f'(E_F) = 0, \tag{4.3}$$
$$f'(x) < 0 \text{ for } x < E_F, f'(x) > 0 \text{ for } x > E_F. \tag{4.4}$$

This leads to

$$E_F = \frac{d_1 b_1 + d_2 b_2}{d_1 + d_2}, \text{ where } d_1 = {a_1}^{\frac{1}{2l-1}} \text{ and } d_2 = {a_2}^{\frac{1}{2l-1}} \tag{4.5}$$

and

$$|I| < \frac{d_1(\frac{c_1}{a_1})^{\frac{1}{2l}} - d_2(\frac{c_2}{a_2})^{\frac{1}{2l}}}{d_1} \text{ or } |I| < \frac{d_2(\frac{c_2}{a_2})^{\frac{1}{2l}} - d_1(\frac{c_1}{a_1})^{\frac{1}{2l}}}{d_1}. \tag{4.6}$$

Since the area below the density function is 1 we get

$$\int_{-(\frac{c}{a})^{\frac{1}{2l}}+b}^{(\frac{c}{a})^{\frac{1}{2l}}+b} c + a(x-b)^{2l} \mathrm{d}x = 1. \tag{4.7}$$

This gives:

$$\Big(\frac{c}{a}\Big)^{\frac{1}{2l}} = \Big(\frac{2l+1}{-4a}\Big)^{\frac{1}{2l+1}}. \tag{4.8}$$

The substitution of Equation (4.8) in (4.6) proves the Theorem. □

Two conclusions can be drawn from Theorem 4.2. First, as the difference between a_1 and a_2 increases, there will be less false positives in the overlap area between the clusters. Figure 4.9(b) illustrates the case for $l = 1$, $a_1 = -0.14$ and $a_2 = -0.002$. Since a_1 (the steep parabola) is smaller than a_2 (the flat parabola), there are no false positives in the overlap area. Second, as the degree of the polynomial increases, the overlap area gets wider and there will be less false positives.

High-Dimensional Parabolas

We generalize Theorem 4.2 to x being a point in a high dimensional space. For simplicity we assume $f(x) = ax^2$.

Theorem 4.3. *[False positives for nD parabolas.] Let n be the dimensionality of the space. Let C_A and C_B be two overlapping clusters. Let $f_1(x)$ and $f_2(x)$ be the density functions of clusters C_A and C_B, respectively:*

$$f_1(x) = \begin{cases} h_1 + a_1\|x - c_1\|^2, \text{ if } \|x - c_1\|^2 \leq \sqrt{-\frac{h_1}{a_1}} \\ 0 \text{ otherwise,} \end{cases}$$

$$f_2(x) = \begin{cases} h_2 + a_2\|x - c_2\|^2, \text{ if } \|x - c_2\|^2 \leq \sqrt{-\frac{h_2}{a_2}} \\ 0 \text{ otherwise,} \end{cases}$$

where c_1 and c_2 are the centroids of the paraboloids, h_1 and h_2 are the maximum heights of the paraboloids. Let I be the overlapping area of the paraboloids. $x \in I$ iff

$$\begin{cases} \|x - c_1\|^2 \leq \sqrt{-\frac{h_1}{a_1}}, \\ \|x - c_2\|^2 \leq \sqrt{-\frac{h_2}{a_2}}. \end{cases}$$

We assume that $I \neq \emptyset, c_1 \notin I, c_2 \notin I$. The overlap area I does not contain false positives iff

$$|I| < \left(\frac{3}{4\pi^{n-1}}\right)^{\frac{1}{3}} \left(\frac{a_i^{\frac{2}{3}} - a_j^{\frac{2}{3}}}{a_j}\right), \tag{4.9}$$

where $i, j \in \{1, 2\} : i \neq j, a_j \geq a_i$ and $|I|$: is the width of the intersection area:

$$|I| = \sqrt{-\frac{h_1}{a_1}} + \sqrt{-\frac{h_2}{a_2}} - \|c_1 - c_2\| \tag{4.10}$$

PROOF: The idea of the proof is based on the observations that false positives can only be on the line connecting E_A and E_B. This observation reduces the case to the one-dimensional case. The application of Theorem 4.2 completes the proof. □

Theorem 4.3 shows that the chances of having false positives decreases as the dimensionality of the dataset increases.

4.8.2 Cores of Higher Dimensionality

The investigation of high dimensional cores can be reduced to the investigation of low dimensional cores with the following reasoning. Let C_1 and C_2 be clusters with curve cores E_1 and E_2. Let f_1 and f_2 be the density functions of the clusters. Let x_1 be a core point in E_1, and x_2 be the closest core point from E_2: $\|x_1 - x_2\| = \min_y \|x_1 - y\|$. Then false positives can only be on the segment of line $[x_1, x_2]$. This reduces the investigation to the one-dimensional case.

4.9 Experiments

This section evaluates CORE clustering experimentally. We organize the experiments into three parts: (i) evaluation of the method for different datasets, (ii) evaluation of the method on real world data, and (iii) comparison with related clustering techniques.

4.9.1 Evaluation for Different Dataset

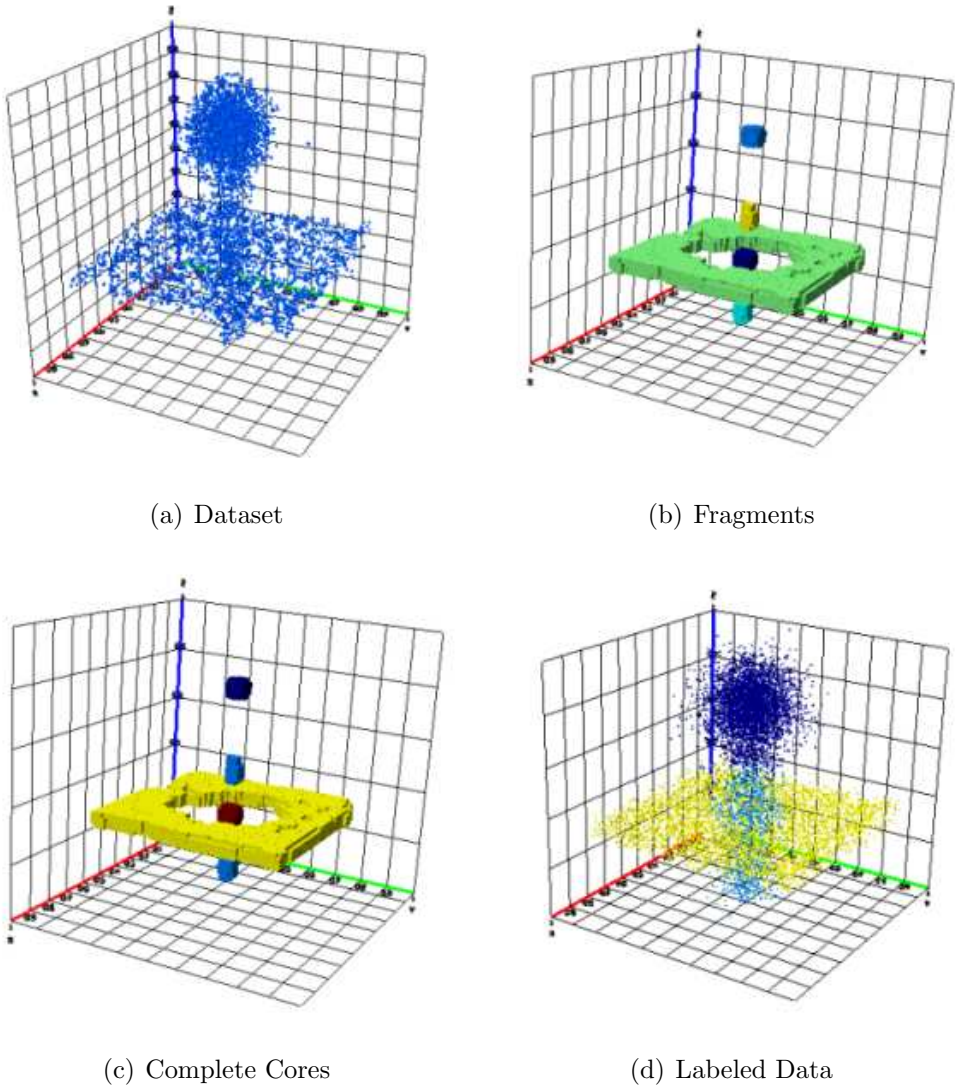
We use synthetic datasets to evaluate the separation of overlapping clusters for different datasets. Specifically, we evaluate the impact of (i) the dimensionality of the cores, (ii) the type of the intersection of cores, and (iii) cores that include each other.

Cores of Different Dimensionality

Figure 4.10 evaluates the impact of the dimensionality of clusters to the clustering quality of CORE for the sphere-line-plane dataset. The dataset consists of three clusters: a sphere (cluster with zero-dimensional core), a line (cluster with one-dimensional core), and a plane (cluster with two-dimensional core). The plane intersects the line and the sphere is located at one end of the line. Although the sphere does not split the line into fragments, the points of the line overlap with the points of the sphere.

Our method clusters the dataset in three steps: it computes fragments and overlaps, reconstructs cores from fragments and overlaps, and labels the data. Figure 4.10(b) shows the computed fragments for the dataset. All fragments are identified correctly. Figure 4.10(c) shows the reconstruction of complete cores from fragments. The two fragments of the line are correctly grouped into a complete core. Finally, Figure 4.10(d) shows the labeling of the data. All data points that have paths pointing to the cores are labeled with the label of the core. The data points with gradient paths pointing to overlaps are distributed between the overlapping cores. Most of the data points in the intersection area are assigned to the sphere, since its density is substantially larger than the density of the line. The data points of the intersection of the plane and the line are equally

distributed between the line and the plane, since the density of the line and the plane are approximately the same.



(a) Dataset (b) Fragments

(c) Complete Cores (d) Labeled Data

Figure 4.10: Intersecting Plane, Line and Sphere

Evaluation of the Intersection Area

In this section we show the invariance of CORE clustering to the type of intersection. Since CORE is invariant wrt the dimensionality of intersecting cores, we only investigate the intersection of one-dimensional cores (curves). We run the following experiments:

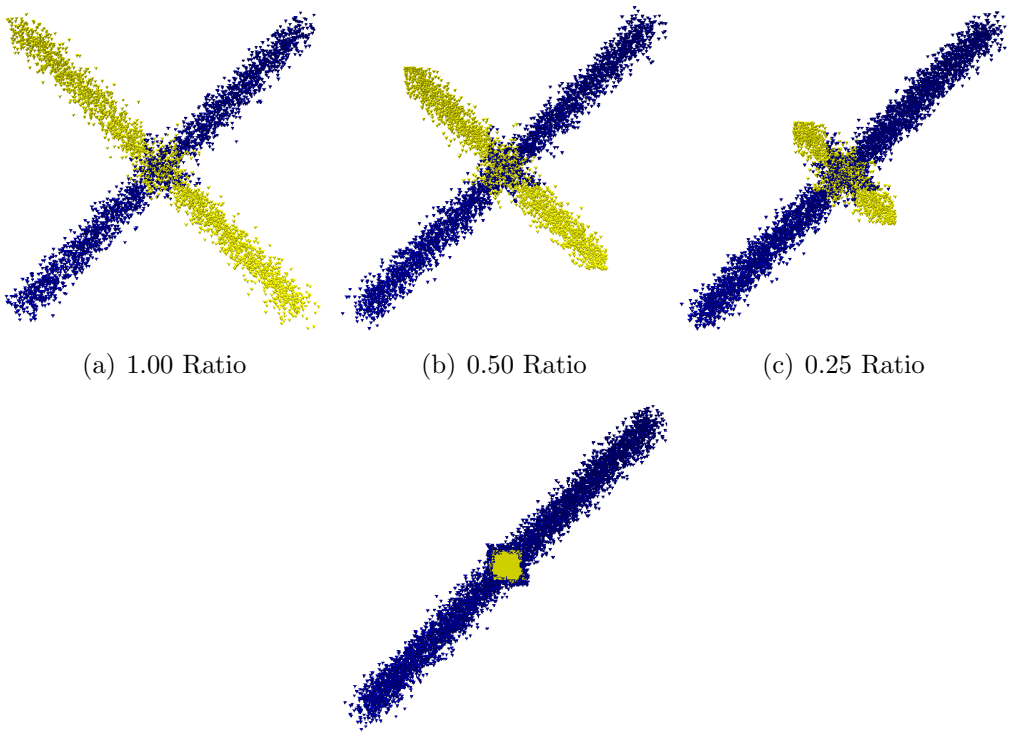
(*i*) varying the ratio of the lengths of intersecting lines.

(*ii*) varying the angle between intersecting lines.

(*iii*) varying the curvature of intersecting curves.

(*iv*) varying the dimensionality of tuples of intersecting lines.

(*i*) Figure 4.11 illustrates CORE clustering for two intersecting lines with varying ratio of the lengths of lines. Clearly, our clustering technique is invariant to the ratio. CORE correctly clusters the data (cf. Figures 4.11(a), 4.11(b) and 4.11(c)). If the length of the shorter line is smaller than the width of the other line, CORE identifies a line and a sphere-core as expected (cf. Figure 4.11(d)).



(a) 1.00 Ratio (b) 0.50 Ratio (c) 0.25 Ratio

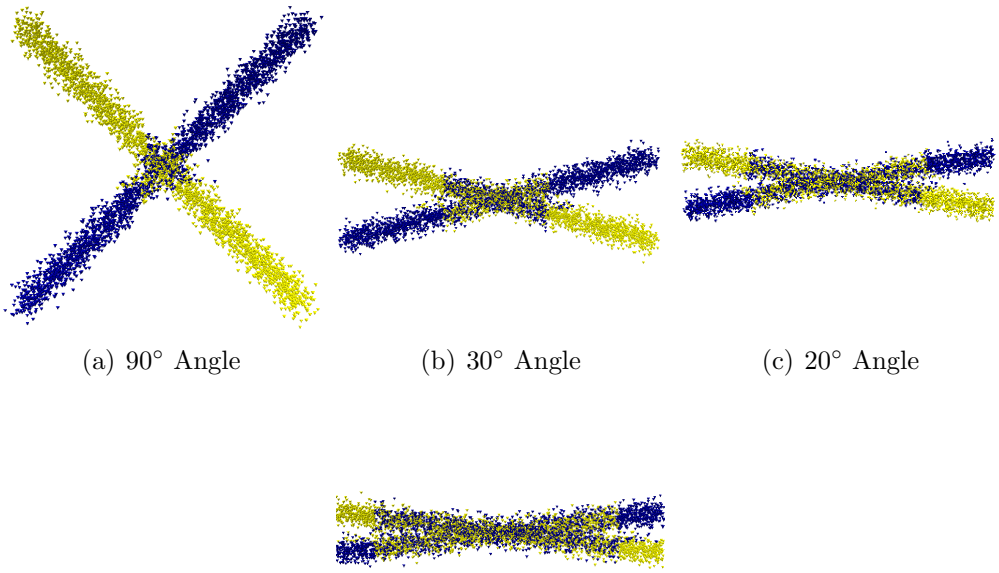
(d) 0.10 Ratio

Figure 4.11: Intersecting Lines for Different Ratios

(*ii*) In Figure 4.12 we evaluate CORE for intersecting lines with varying angles between the lines. Our clustering method correctly identified cores and overlaps of clusters for angles up to 15°. As the angle of the intersecting lines decreases, the overlap area of the cores increases.

(*iii*) Figure 4.13 evaluates the impact of the curvature of intersecting clusters. In this experiment we intersect a line with a circle with a different radius. As the radius of the circle decreases, the curvature of the curve increases. Our clustering technique correctly identifies clusters independent of whether the objects are straight lines or curves.

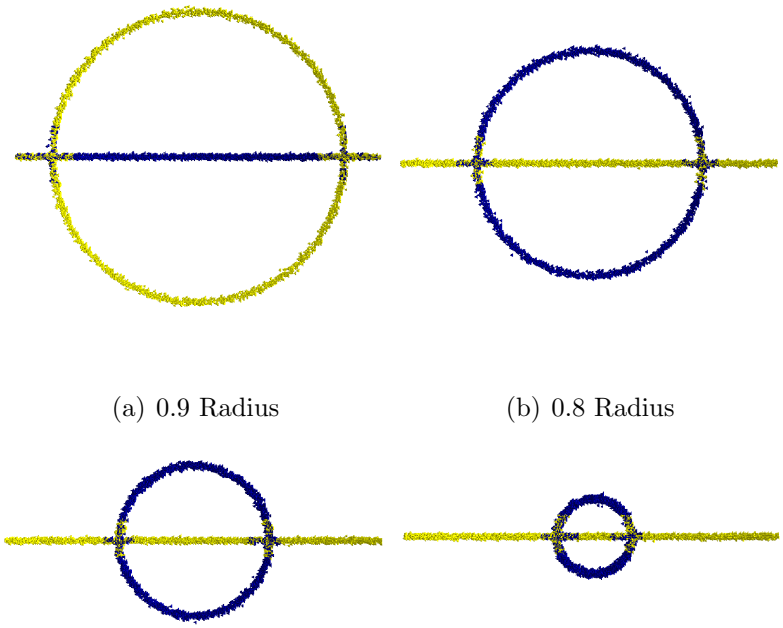
(*iv*) Because CORE is a grid based clustering technique, it may happen that for highly overlapping clusters CORE identifies an overlap with a larger area than



(a) 90° Angle (b) 30° Angle (c) 20° Angle

(d) 15° Angle

Figure 4.12: Intersecting Lines for Different Angles



(a) 0.9 Radius (b) 0.8 Radius

(c) 0.5 Radius (d) 0.2 Radius

Figure 4.13: Intersecting Curve-Core Clusters for Different Curvatures

it is in reality. As an example of such a behavior, consider two intersecting lines with 15° angle between them (cf. Figure 4.12(d). Since the lines highly overlap, data tuples that are close to the overlap are assigned to the overlap. However, the correctness of labeling data tuples depends only on the size of an overlap, but not on the curvature of overlapping cluster or dimensionality of the data. Table 4.1 lists a percentage of correctly labeled tuples for intersecting lines while varying the dimensionality of the data. Clearly, since the percentage remains the same for any number of dimensions, CORE is invariant to the dimensionality of the data.

Dimensionality	Number of Correctly Labeled Tuples
2	96.75 %
3	96.66 %
4	96.83 %
5	96.78 %
6	96.55 %
7	96.67 %

Table 4.1: The Intersection Area in High Dimensional Data.

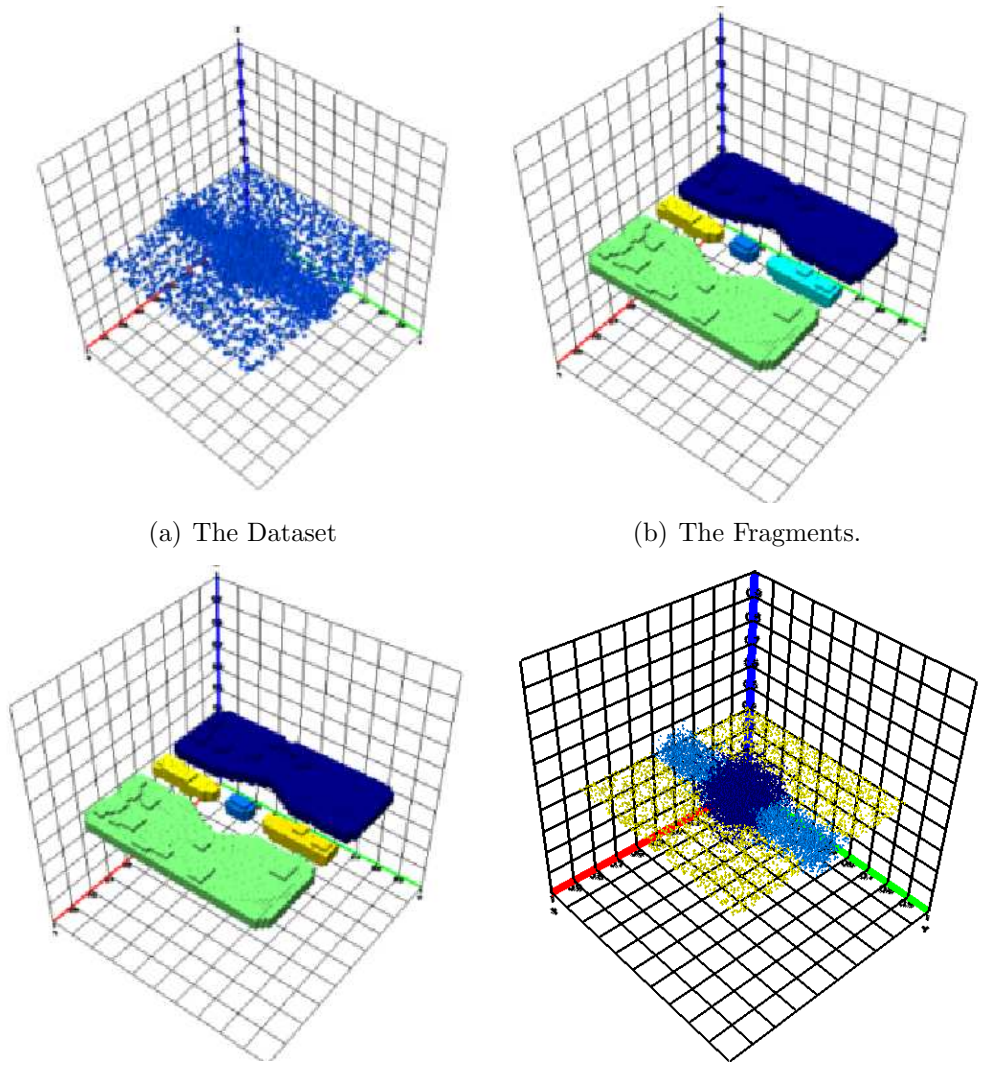
Clusters Containing other Clusters

Figure 4.14 evaluates CORE for clusters containing each other. There are three structures in the data: a sphere with a core that is enclosed by the core of a line, and a plane with a core that contains the cores of the sphere and the line. Core separates the clusters correctly: it finds five fragments and combines them into three complete cores. Figure 4.14(d) shows the achieved labeling with three clusters.

4.9.2 Real World Data

In this section we continue the experiment with web logs and financial data that was introduced in Section 3.9. This type of transactional data is very common and records activities over a continuous period of time. For web logs a click on a web page is the activity that is recorded in the database. For finantial data each payment is stored in the database. With each activity a number of attributes are stored, such as the URL, account ID, etc.

Transactional data is an interesting application area for clustering techniques, since the clustering of activities helps to get a better business understanding.



(a) The Dataset (b) The Fragments.

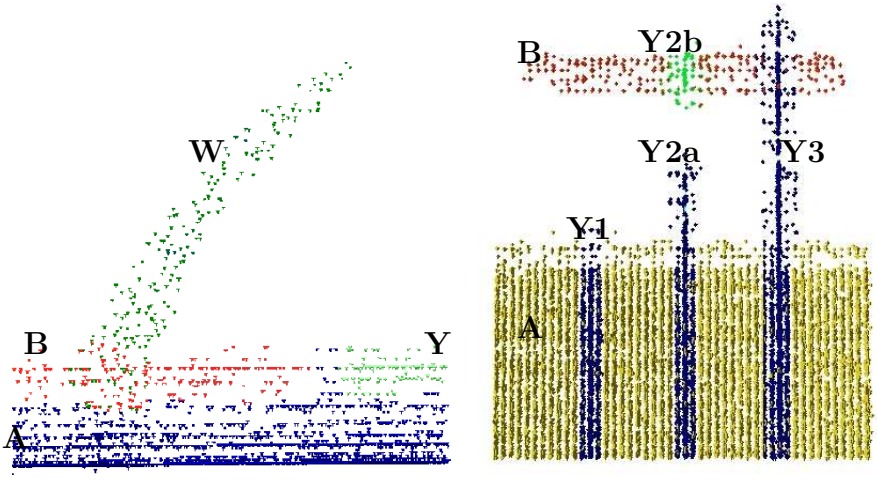
(c) The Reconstructed Cores. (d) Labeled Data.

Figure 4.14: Intersection of Plane, Line and Sphere

A key issue when analyzing transactional data is the often extreme skew. Skew arises because of very high volume activities that dominate all other activities and because of strong behavioral patterns (bank transaction at the end of the fiscal year; clicks on high-frequency portal pages). In order to investigate highly skewed data it is often necessary to identify dominant clusters and remove them.

In Figure 4.15 we present experiments that evaluate the potential of CORE to identify and separate overlapping clusters in time-varying data. Figure 4.15(a) shows that CORE successfully separates clicks produced by search robots and restores original cluster (cluster B in is restored from fragments B1 and B2 in Figure 3.7(b)). Figure 4.15(b) shows that CORE successfully separates end-of-the-year transaction from other data and restores original clusters (clusters A

and B are restored from fragments A1, A2, A3, A4, B1, B2, B3 in Figure 3.7(d)). CORE identified all the fragments and reconstructed the cores correctly.



(a) ClickStream Data (b) Financial Data

Figure 4.15: Separation of Overlapping Clusters in Real World Datasets

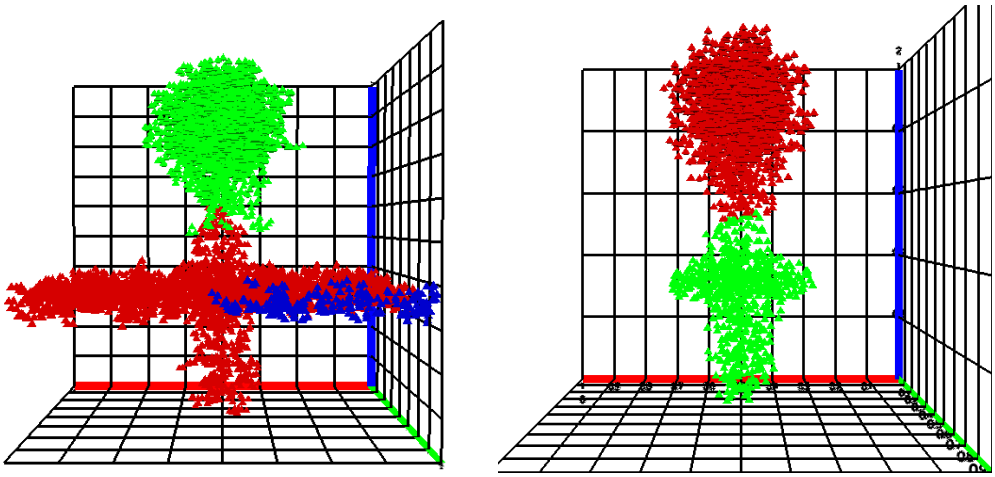
4.9.3 Comparison with CURE and DBScan

In this section we compare our method with CURE and DBScan clustering techniques on datasets with overlapping clusters. CURE and DBScan are well known and are good representatives of hierarchical (CURE) and density based (DBScan) clustering methods. Our experimental studies in Section 3.9 show that CURE and DBScan produce better quality clusters than DataBubbles, K-Means and RIC on datasets with arbitrary shaped clusters and/or with clusters which are close to each other. Note, that in contrast to our method, all the above mentioned clustering techniques identify clusters only at one density level and, hence, are not capable to separate overlapping clusters.

We use the sphere-line-plane dataset to show how other clustering techniques deal with overlapping clusters. Remember, that CORE correctly identifies the cores of the clusters and labeled the data properly (cf. Figure 4.10).

CURE employs two techniques to cluster the data: (i) representative points and (ii) shrinkage of representative towards the center of the cluster. Representative points enable CURE to find clusters of arbitrary shape, while the shrinkage of representative points towards the center makes the clustering robust wrt anomalies of the shape of clusters. The method needs three parameters: number of clusters k, number of representative points m, and shrinkage level α.

We ran CURE clustering with many different input parameters. Figure 4.16(a) illustrates one of the best outputs of the CURE clustering (the input parameters are: $k = 3$, $m = 20$, $\alpha = 0.3$). CURE clustering successfully identifies the



(a) CURE (b) DBScan

Figure 4.16: Other Clustering Methods

sphere (since it is a spherical structure), and most of the plane (due to a good allocation of representative points). CURE fails to identify the line cluster and a part of the plane becomes a separate cluster. CURE misses the line because there are two intersection areas of high density very close to the line. Therefore the representative points are pulled towards the intersection areas and are merged either with the plane or with the sphere. Finally, since it tries to find 3 clusters, it allocates the third cluster to the most distant corner of the plane. Other values of m and α do not increase the quality of the clustering. Intuitively, the negative impact of the overlapping areas of the clusters can be decreased by decreasing the shrinkage parameter α. However, this worsens the quality, since the representative points are placed randomly in space, rather than on the core of the clusters. Increasing the representative points also does not help, since most of them are pulled towards the overlaps of the clusters.

The typical output of the DBScan algorithm is illustrated in Figure 4.16(b). The algorithm depends on two parameters: the size of the neighborhood ε and the number of points in the neighborhood $minPts$. DBScan correctly identifies the sphere cluster and most of the line. Since the density of the plane is slightly lower than the density of line, the plane cluster is filtered out as noise. If the input parameters of the algorithm are adjusted, such that the plane is not filtered as noise, all three clusters are merged together and only one cluster is found.

4.10 Conclusions and Future work

In this chapter we extend CORE to separate overlapping clusters. Cores of overlapping clusters are divided into overlaps and fragments. We reconstruct complete cores in two steps: first, we classify cores into fragments and overlaps and, next,

we connect fragments that are separated by the overlap based on their density level and orientation. We provide an extensive evaluation of our technique for synthetic and real world data. The experiments demonstrate the invariance wrt the dimensionality of the clusters, and the angle and curvature of the overlapping structures.

In the future we want to investigate the trade-off between shape and density information for reconstructing cores from fragments. CORE connects fragments if they match in terms of shape and density. It is possible to weight these factors differently and, e.g., connect fragments based on their shape only. Finally, the labeling of data points should be refined. For the labeling of the data points we used a randomized model that is based on overlaps. By using the cores it is possible to improve the labeling in overlapping areas.

Chapter 5

Summary of Conclusions

The Ph.D. thesis makes three main contributions: it develops a hierarchical data summary structure for multidimensional data that requires minimal intermediate memory, it uses the data summary structure to cluster multidimensional data without requiring any input parameters, and it proposes a novel clustering technique that identifies and separates overlapping clusters.

Data summary structures are heavily used in query optimization and approximate answering of aggregate range queries. The construction of data summary structures is limited to low dimensional data due to the expensive requirements in terms of intermediate memory. We develop the AD-Tree, a hierarchical data summary structure that summarizes multidimensional data in terms of its density and can be constructed without additional intermediate memory. The construction of the AD-Tree starts from a sparse initial grid whose cells are iteratively split in places and along dimension where the density function of the data is non-linear. This guarantees a minimal usage of intermediate memory since the split places new grid points only in areas where required to increase the precision of the density estimation. We define and implement the split with the help of the *shape error* and *dimensional split*. The shape error measures the deviation of the estimated density function from being linear on a one-dimensional grid. The AD-Tree approximates the shape error by considering the density information stored in the AD-Tree. The dimensional split extends the shape error to multidimensional grids: it identifies cells that exhibit a high shape error and splits the cells along dimensions where the shape error is too high. We optimize the size of the AD-Tree for multidimensional data. We minimize the number of grids in the AD-Tree by grouping small grids into grids of maximal volume. The compact representation of multidimensional grids reduces the cost to store them by a factor of the dimensionality d. We analytically investigate the properties of the AD-Tree: we prove optimality of the AD-Tree with respect to the initial partition, and we show that the AD-Tree adjusts to the local dimensionality of structures in the data and robustly approximates the shape error if the initial

partition is chosen based on extrema points of the density. The experiments on real world and synthetics datasets confirm the analytical results.

In order to precisely compute aggregate range queries on the AD-Tree we must efficiently compute multiple integrals and eliminate overlap between grids on different levels of the hierarchy. The thesis offers an effective solution for this. First, we replace the expensive computation of multiple integrals by a cheap linear interpolation. We prove that our approach to compute multiple integrals is exact. Next, we develop algorithms to filter out overlaps between grids in the query range. We compare the AD-Tree with the state of the art techniques and validate the effectiveness of our approach with experiments on real world data.

The performance and quality of clustering techniques depends on carefully chosen input parameters. It is hard and often time-consuming to choose these parameters. The thesis develops CORE, a clustering technique that computes clusters without any input parameters. CORE is a nonparametric clustering and it explicitly computates and represents local maxima. The AD-Tree, which provides a compact and uniform estimate of the density, enables the efficient computation of the local maxima. CORE represents local maxima with core: connected set of grid points in the AD-Tree that approximates local maxima of the density. We robustly compute cores with the help of rectangular neighborhoods and gradients. A rectangular neighborhood localizes stationary points in the AD-Tree and gradients distinguish local maxima from other types of stationary points. We analytically prove the correctness of CORE. Our experimental investigation compares CORE with sate of the art clustering techniques on various synthetic and real world datasets. CORE is as fast as other techniques and outperforms them in terms of the quality of computed clusters.

Separation of overlapping clusters arises in many real world datasets and is especially common to time-varying data. The separation of overlapping clusters is difficult for existing clustering techniques since they are either limited to find clusters on one density level and/or their model of clusters does not consider the orientation of clusters. CORE overcomes these limitations and correctly separates overlapping clusters. The cores of overlapping clusters divide each over into fragments and overlaps. We use the AD-Tree to find fragments and overlaps of cores at all density levels. We restore complete cores from their fragments in two steps. First, with use gradient path to assign fragments to overlaps and compute their orientation. Next, we restore complete cores from fragments that are assigned to the same overlap and are similarly oriented. We analytically investigate the conditions for a successful separation of overlapping clusters with polynomial density function. We prove the effectiveness of our approach on synthetics datasets with various properties of overlapping clusters, as well, as on real world datasets.

We extensively evaluate our methods on real world and synthetics dataset both visually and numerically. Our collected experience resulted on the XVDM, a new framework for visual data mining, which gives the ability to compare, investigate and learn different data mining techniques. The existing approaches are focused

mostly on the visualization of the results and it is difficult to use them for the investigation of data mining techniques by slightly changing the input dataset or comparing results for different parameters. The salient properties if the XVDM are: it enables construction of synthetics datasets in a real time and from the number of primitives; it supports unlimited number of data mining techniques to be run at the same time; it provides an intuitive interface to switch between different techniques and compare the results of them numerically or visually. The XVDM is successfully used by students in Free University of Bolzano-Bozen.

Bibliography

[1] Acharya, S., Poosala, V., and Ramaswamy, S. 1999. Selectivity estimation in spatial databases. In Proceedings of the 1999 ACM SIGMOD international Conference on Management of Data (Philadelphia, Pennsylvania, United States, May 31 - June 03, 1999). SIGMOD '99. ACM, New York, NY, 13-24. DOI= http://doi.acm.org/10.1145/304182.304184

[2] Agrawal, R., Gehrke, J., Gunopulos, D., and Raghavan, P. 1998. Automatic subspace clustering of high dimensional data for data mining applications. SIGMOD Rec. 27, 2 (Jun. 1998), 94-105. DOI= http://doi.acm.org/10.1145/276305.276314

[3] Ankerst, M., Breunig, M. M., Kriegel, H., and Sander, J. 1999. OPTICS: ordering points to identify the clustering structure. In Proceedings of the 1999 ACM SIGMOD international Conference on Management of Data (Philadelphia, Pennsylvania, United States, May 31 - June 03, 1999). SIGMOD '99. ACM, New York, NY, 49-60. DOI= http://doi.acm.org/10.1145/304182.304187

[4] Babcock, B., Chaudhuri, S., and Das, G. 2003. Dynamic sample selection for approximate query processing. In Proceedings of the 2003 ACM SIGMOD international Conference on Management of Data (San Diego, California, June 09 - 12, 2003). SIGMOD '03. ACM, New York, NY, 539-550. DOI= http://doi.acm.org/10.1145/872757.872822

[5] Bradley, P. S. and Fayyad, U. M. 1998. Refining Initial Points for K-Means Clustering. In Proceedings of the Fifteenth international Conference on Machine Learning (July 24 - 27, 1998). J. W. Shavlik, Ed. Morgan Kaufmann Publishers, San Francisco, CA, 91-99.

[6] Bruno, N., Chaudhuri, S., and Gravano, L. 2001. STHoles: a multidimensional workload-aware histogram. In Proceedings of the 2001 ACM SIGMOD international Conference on Management of Data (Santa Barbara, California, United States, May 21 - 24, 2001). T. Sellis, Ed. SIGMOD '01. ACM, New York, NY, 211-222. DOI= http://doi.acm.org/10.1145/375663.375686

[7] Böhm, C., Faloutsos, C., Pan, J., and Plant, C. 2007. RIC: Parameter-free noise-robust clustering. ACM Trans. Knowl. Discov. Data 1, 3 (Dec. 2007), 10. DOI= http://doi.acm.org/10.1145/1297332.1297334

[8] Böhm, C., Kailing, K., Kröger, P., and Zimek, A. 2004. Computing Clusters of Correlation Connected objects. In Proceedings of the 2004 ACM SIGMOD international Conference on Management of Data (Paris, France, June 13 - 18, 2004). SIGMOD '04. ACM, New York, NY, 455-466. DOI= http://doi.acm.org/10.1145/1007568.1007620

[9] Breunig, M. M., Kriegel, H., Krger, P., and Sander, J. 2001. Data bubbles: quality preserving performance boosting for hierarchical clustering. In Proceedings of the 2001 ACM SIGMOD international Conference on Management of Data (Santa Barbara, California, United States, May 21 - 24, 2001). T. Sellis, Ed. SIGMOD '01. ACM, New York, NY, 79-90. DOI= http://doi.acm.org/10.1145/375663.375672

[10] Cao F. and Ester M. and W. Qian and A. Zhou. 2006. Density-Based Clustering over an Evolving Data Stream with Noise. In SIAM International Conference on Data Mining, 2006.

[11] Chakrabarti, D., Kumar, R., and Tomkins, A. 2006. Evolutionary clustering. In Proceedings of the 12th ACM SIGKDD international Conference on Knowledge Discovery and Data Mining (Philadelphia, PA, USA, August 20 - 23, 2006). KDD '06. ACM, New York, NY, 554-560. DOI= http://doi.acm.org/10.1145/1150402.1150467

[12] Chaudhuri, S., Das, G., and Narasayya, V. 2001. A robust, optimization-based approach for approximate answering of aggregate queries. In Proceedings of the 2001 ACM SIGMOD international Conference on Management of Data (Santa Barbara, California, United States, May 21 - 24, 2001). T. Sellis, Ed. SIGMOD '01. ACM, New York, NY, 295-306. DOI= http://doi.acm.org/10.1145/375663.375694

[13] Chen, Y. and Tu, L. 2007. Density-based clustering for real-time stream data. In Proceedings of the 13th ACM SIGKDD international Conference on Knowledge Discovery and Data Mining (San Jose, California, USA, August 12 - 15, 2007). KDD '07. ACM, New York, NY, 133-142. DOI= http://doi.acm.org/10.1145/1281192.1281210

[14] Davidson, I., Ravi, S. S., and Ester, M. 2007. Efficient incremental constrained clustering. In Proceedings of the 13th ACM SIGKDD international Conference on Knowledge Discovery and Data Mining (San Jose, California, USA, August 12 - 15, 2007). KDD '07. ACM, New York, NY, 240-249. DOI= http://doi.acm.org/10.1145/1281192.1281221

[15] Deligiannakis, A., Garofalakis, M., and Roussopoulos, N. 2007. Extended wavelets for multiple measures. ACM Trans. Database Syst. 32, 2 (Jun. 2007), 10. DOI= http://doi.acm.org/10.1145/1242524.1242527

[16] Deshpande, A., Garofalakis, M., and Rastogi, R. 2001. Independence is good: dependency-based histogram synopses for high-dimensional data. SIGMOD Rec. 30, 2 (Jun. 2001), 199-210. DOI= http://doi.acm.org/10.1145/376284.375685

[17] Ester, M., H.-P. Kriegel, J. Sander, X. Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In Proceedings of Second International Conference on Knowledge Discovery and Data Mining, Portland, USA, pp. 226-231.

[18] Ganti, V., Lee, M., and Ramakrishnan, R. 2000. ICICLES: Self-Tuning Samples for Approximate Query Answering. In Proceedings of the 26th international Conference on Very Large Data Bases (September 10 - 14, 2000). A. E. Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K. Whang, Eds. Very Large Data Bases. Morgan Kaufmann Publishers, San Francisco, CA, 176-187.

[19] Garofalakis, M. and Kumar, A. 2004. Deterministic wavelet thresholding for maximum-error metrics. In Proceedings of the Twenty-Third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (Paris, France, June 14 - 16, 2004). PODS '04. ACM, New York, NY, 166-176. DOI= http://doi.acm.org/10.1145/1055558.1055582

[20] Gemulla, R., Lehner, W., and Haas, P. J. 2007. Maintaining bernoulli samples over evolving multisets. In Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (Beijing, China, June 11 - 13, 2007). PODS '07. ACM, New York, NY, 93-102. DOI= http://doi.acm.org/10.1145/1265530.1265544

[21] Gibbons, P. B. and Matias, Y. 1998. New sampling-based summary statistics for improving approximate query answers. In Proceedings of the 1998 ACM SIGMOD international Conference on Management of Data (Seattle, Washington, United States, June 01 - 04, 1998). A. Tiwary and M. Franklin, Eds. SIGMOD '98. ACM, New York, NY, 331-342. DOI= http://doi.acm.org/10.1145/276304.276334

[22] Gionis, A., Mannila, H., and Tsaparas, P. 2007. Clustering aggregation. ACM Trans. Knowl. Discov. Data 1, 1 (Mar. 2007), 4. DOI= http://doi.acm.org/10.1145/1217299.1217303

[23] Guha, S. 2005. Space efficiency in synopsis construction algorithms. In Proceedings of the 31st international Conference on Very Large Data Bases (Trondheim, Norway, August 30 - September 02, 2005). Very Large Data Bases. VLDB Endowment, 409-420.

[24] Guha, S., Rastogi, R., and Shim, K. 1998. CURE: an efficient clustering algorithm for large databases. In Proceedings of the 1998 ACM SIGMOD international Conference on Management of Data (Seattle, Washington, United States, June

01 - 04, 1998). A. Tiwary and M. Franklin, Eds. SIGMOD '98. ACM, New York, NY, 73-84. DOI= http://doi.acm.org/10.1145/276304.276312

[25] Guha, S., Shim, K., and Woo, J. 2004. REHIST: relative error histogram construction algorithms. In Proceedings of the Thirtieth international Conference on Very Large Data Bases - Volume 30 (Toronto, Canada, August 31 - September 03, 2004). M. A. Nascimento, M. T. zsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, Eds. Very Large Data Bases. VLDB Endowment, 300-311.

[26] Gunopulos, D., Kollios, G., Tsotras, J., and Domeniconi, C. 2005. Selectivity estimators for multidimensional range queries over real attributes. The VLDB Journal 14, 2 (Apr. 2005), 137-154. DOI= http://dx.doi.org/10.1007/s00778-003-0090-4

[27] Heinz, C. and Seeger, B. 2007. Adaptive Wavelet Density Estimators over Data Streams. In Proceedings of the 19th international Conference on Scientific and Statistical Database Management (July 09 - 11, 2007). SSDBM. IEEE Computer Society, Washington, DC, 35. DOI= http://dx.doi.org/10.1109/SSDBM.2007.28

[28] Hinneburg A. and D. Keim. An efficient approach to clustering in large multimedia databases with noise. In Proc. 4th Int. Conf. on Knowledge Discovery and Data Minging, pages 58–65, 1998.

[29] Ioannidis, Y. 2003. The history of histograms (abridged). In Proceedings of the 29th international Conference on Very Large Data Bases - Volume 29 (Berlin, Germany, September 09 - 12, 2003). J. C. Freytag, P. C. Lockemann, S. Abiteboul, M. J. Carey, P. G. Selinger, and A. Heuer, Eds. Very Large Data Bases. VLDB Endowment, 19-30.

[30] Ioannidis, Y. E. and Poosala, V. 1995. Balancing histogram optimality and practicality for query result size estimation. In Proceedings of the 1995 ACM SIGMOD international Conference on Management of Data (San Jose, California, United States, May 22 - 25, 1995). M. Carey and D. Schneider, Eds. SIGMOD '95. ACM, New York, NY, 233-244. DOI= http://doi.acm.org/10.1145/223784.223841

[31] Jagadish, H. V., Koudas, N., Muthukrishnan, S., Poosala, V., Sevcik, K. C., and Suel, T. 1998. Optimal Histograms with Quality Guarantees. In Proceedings of the 24rd international Conference on Very Large Data Bases (August 24 - 27, 1998). A. Gupta, O. Shmueli, and J. Widom, Eds. Very Large Data Bases. Morgan Kaufmann Publishers, San Francisco, CA, 275-286.

[32] Jermaine, C. 2003. Robust estimation with sampling and approximate preaggregation. In Proceedings of the 29th international Conference on Very Large Data Bases - Volume 29 (Berlin, Germany, September 09 - 12, 2003). J. C. Freytag, P. C. Lockemann, S. Abiteboul, M. J. Carey, P. G. Selinger, and A. Heuer, Eds. Very Large Data Bases. VLDB Endowment, 886-897.

[33] Jiang, D., Eick, C. F., and Chen, C. 2007. On supervised density estimation techniques and their application to spatial data mining. In Proceedings of the 15th Annual ACM international Symposium on Advances in Geographic information Systems (Seattle, Washington, November 07 - 09, 2007). GIS '07. ACM, New York, NY, 1-4. DOI= http://doi.acm.org/10.1145/1341012.1341089

[34] Kailing Karin and Kriegel Hans-peter and Krger Peer. 2004. Density-connected subspace clustering for high-dimensional data. In SIAM International Conference on Data Mining, 246–257.

[35] Karras, P. and Mamoulis, N. 2005. One-pass wavelet synopses for maximum-error metrics. In Proceedings of the 31st international Conference on Very Large Data Bases (Trondheim, Norway, August 30 - September 02, 2005). Very Large Data Bases. VLDB Endowment, 421-432.

[36] Knuth, Donald E. 2005. The Art of Computer Programming, Volume 1. ISBN 0201853922. Addison-Wesley Professional.

[37] Kriegel, H. and Pfeifle, M. 2005. Density-based clustering of uncertain data. In Proceedings of the Eleventh ACM SIGKDD international Conference on Knowledge Discovery in Data Mining (Chicago, Illinois, USA, August 21 - 24, 2005). KDD '05. ACM, New York, NY, 672-677. DOI= http://doi.acm.org/10.1145/1081870.1081955

[38] Kriegel, H. and Pfeifle, M. 2005. Hierarchical Density-Based Clustering of Uncertain Data. In Proceedings of the Fifth IEEE international Conference on Data Mining (November 27 - 30, 2005). ICDM. IEEE Computer Society, Washington, DC, 689-692. DOI= http://dx.doi.org/10.1109/ICDM.2005.75

[39] Lazaridis, I. and Mehrotra, S. 2001. Progressive approximate aggregate queries with a multi-resolution tree structure. In Proceedings of the 2001 ACM SIGMOD international Conference on Management of Data (Santa Barbara, California, United States, May 21 - 24, 2001). T. Sellis, Ed. SIGMOD '01. ACM, New York, NY, 401-412. DOI= http://doi.acm.org/10.1145/375663.375718

[40] Li, T. 2005. A general model for clustering binary data. In Proceedings of the Eleventh ACM SIGKDD international Conference on Knowledge Discovery in Data Mining (Chicago, Illinois, USA, August 21 - 24, 2005). KDD '05. ACM, New York, NY, 188-197. DOI= http://doi.acm.org/10.1145/1081870.1081894

[41] Liu, J., Strohmaier, K., and Wang, W. 2004. Revealing True Subspace Clusters in High Dimensions. In Proceedings of the Fourth IEEE international Conference on Data Mining (November 01 - 04, 2004). ICDM. IEEE Computer Society, Washington, DC, 463-466.

[42] Luo, J., Zhou, X., Zhang, Y., Shen, H. T., and Li, J. 2007. Selectivity estimation by batch-query based histogram and parametric method. In Proceedings of the

Eighteenth Conference on Australasian Database - Volume 63 (Ballarat, Victoria, Australia, January 30 - February 02, 2007). J. Bailey and A. Fekete, Eds. ACM International Conference Proceeding Series, vol. 242. Australian Computer Society, Darlinghurst, Australia, 93-102.

[43] Matias, Y., Vitter, J. S., and Wang, M. 1998. Wavelet-based histograms for selectivity estimation. SIGMOD Rec. 27, 2 (Jun. 1998), 448-459. DOI= http://doi.acm.org/10.1145/276305.276344

[44] Mazeika, A., Böhlen, M. H., and Taliun, A. 2006. Adaptive density estimation. In Proceedings of the 32nd international Conference on Very Large Data Bases (Seoul, Korea, September 12 - 15, 2006). U. Dayal, K. Whang, D. Lomet, G. Alonso, G. Lohman, M. Kersten, S. K. Cha, and Y. Kim, Eds. Very Large Data Bases. VLDB Endowment, 1191-1194.

[45] Moise, G. and Sander, J. 2008. Finding non-redundant, statistically significant regions in high dimensional data: a novel approach to projected and subspace clustering. In Proceeding of the 14th ACM SIGKDD international Conference on Knowledge Discovery and Data Mining (Las Vegas, Nevada, USA, August 24 - 27, 2008). KDD '08. ACM, New York, NY, 533-541. DOI= http://doi.acm.org/10.1145/1401890.1401956

[46] Muralikrishna, M. and DeWitt, D.J. 1988. Equidepth histograms for estimating selectivity factors for multi-dimensional queries. Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 28-36, Chicago, Illinois.

[47] Nath, S. and Gibbons, P. B. 2008. Online maintenance of very large random samples on flash storage. Proc. VLDB Endow. 1, 1 (Aug. 2008), 970-983. DOI= http://doi.acm.org/10.1145/1453856.1453961

[48] Ng, R. T. and Han, J. 1994. Efficient and Effective Clustering Methods for Spatial Data Mining. In Proceedings of the 20th international Conference on Very Large Data Bases (September 12 - 15, 1994). J. B. Bocca, M. Jarke, and C. Zaniolo, Eds. Very Large Data Bases. Morgan Kaufmann Publishers, San Francisco, CA, 144-155.

[49] Pelleg, D. and Moore, A. W. 2000. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In Proceedings of the Seventeenth international Conference on Machine Learning (June 29 - July 02, 2000). P. Langley, Ed. Morgan Kaufmann Publishers, San Francisco, CA, 727-734.

[50] Poosala, V. and Ioannidis, Y. E. 1997. Selectivity Estimation Without the Attribute Value Independence Assumption. In Proceedings of the 23rd international Conference on Very Large Data Bases (August 25 - 29, 1997). M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, Eds. Very Large Data Bases. Morgan Kaufmann Publishers, San Francisco, CA, 486-495.

[51] Redmond, S. J. and Heneghan, C. 2007. A method for initialising the K-means clustering algorithm using kd-trees. Pattern Recogn. Lett. 28, 8 (Jun. 2007), 965-973. DOI= http://dx.doi.org/10.1016/j.patrec.2007.01.001

[52] Reiss, F., Garofalakis, M., and Hellerstein, J. M. 2006. Compact histograms for hierarchical identifiers. In Proceedings of the 32nd international Conference on Very Large Data Bases (Seoul, Korea, September 12 - 15, 2006). U. Dayal, K. Whang, D. Lomet, G. Alonso, G. Lohman, M. Kersten, S. K. Cha, and Y. Kim, Eds. Very Large Data Bases. VLDB Endowment, 870-881.

[53] Scott D. W., 1992, Multivariate Density Estimation, Wiley & Sons, New York, 317.

[54] Sheikholeslami, G., Chatterjee, S., and Zhang, A. 2000. WaveCluster: a wavelet-based clustering approach for spatial data in very large databases. The VLDB Journal 8, 3-4 (Feb. 2000), 289-304. DOI= http://dx.doi.org/10.1007/s0077

[55] Shen, Y., Shen, Z., Zhang, S., and Yang, Q. 2004. Cluster Cores-Based Clustering for High Dimensional Data. In Proceedings of the Fourth IEEE international Conference on Data Mining (November 01 - 04, 2004). ICDM. IEEE Computer Society, Washington, DC, 519-522.

[56] Shi, Y., Song, Y., and Zhang, A. 2003. A shrinking-based approach for multi-dimensional data analysis. In Proceedings of the 29th international Conference on Very Large Data Bases - Volume 29 (Berlin, Germany, September 09 - 12, 2003). J. C. Freytag, P. C. Lockemann, S. Abiteboul, M. J. Carey, P. G. Selinger, and A. Heuer, Eds. Very Large Data Bases. VLDB Endowment, 440-451.

[57] Silverman B.W., 1986, Density Estimation for Statistics and Data Analysis, Chapman & Hall, London, 175+ix.

[58] Thaper, N., Guha, S., Indyk, P., and Koudas, N. 2002. Dynamic multidimensional histograms. In Proceedings of the 2002 ACM SIGMOD international Conference on Management of Data (Madison, Wisconsin, June 03 - 06, 2002). SIGMOD '02. ACM, New York, NY, 428-439. DOI= http://doi.acm.org/10.1145/564691.564741

[59] Tung, A. K., Xu, X., and Ooi, B. C. 2005. CURLER: finding and visualizing nonlinear correlation clusters. In Proceedings of the 2005 ACM SIGMOD international Conference on Management of Data (Baltimore, Maryland, June 14 - 16, 2005). SIGMOD '05. ACM, New York, NY, 467-478. DOI= http://doi.acm.org/10.1145/1066157.1066211

[60] Vitter, J. S., Wang, M., and Iyer, B. 1998. Data cube approximation and histograms via wavelets. In Proceedings of the Seventh international Conference on information and Knowledge Management (Bethesda, Maryland, United States, November 02 - 07, 1998). CIKM '98. ACM, New York, NY, 96-104. DOI= http://doi.acm.org/10.1145/288627.288645

[61] Wang, H. and Sevcik, K. C. 2003. A multi-dimensional histogram for selectivity estimation and fast approximate query answering. In Proceedings of the 2003 Conference of the Centre For Advanced Studies on Collaborative Research (Toronto, Ontario, Canada, October 06 - 09, 2003). IBM Centre for Advanced Studies Conference. IBM Press, 328-342.

[62] Wang, H. and Sevcik, K. C. 2008. Histograms based on the minimum description length principle. The VLDB Journal 17, 3 (May. 2008), 419-442. DOI= http://dx.doi.org/10.1007/s00778-006-0015-0

[63] Wang, H., Wang, W., Yang, J., and Yu, P. S. 2002. Clustering by pattern similarity in large data sets. In Proceedings of the 2002 ACM SIGMOD international Conference on Management of Data (Madison, Wisconsin, June 03 - 06, 2002). SIGMOD '02. ACM, New York, NY, 394-405. DOI= http://doi.acm.org/10.1145/564691.564737

[64] Wu, J., Xiong, H., and Chen, J. 2008. SAIL: summation-based incremental learning for information-theoretic clustering. In Proceeding of the 14th ACM SIGKDD international Conference on Knowledge Discovery and Data Mining (Las Vegas, Nevada, USA, August 24 - 27, 2008). KDD '08. ACM, New York, NY, 740-748. DOI= http://doi.acm.org/10.1145/1401890.1401979

[65] Xu, X., Ester, M., Kriegel, H., and Sander, J. 1998. A Distribution-Based Clustering Algorithm for Mining in Large Spatial Databases. In Proceedings of the Fourteenth international Conference on Data Engineering (February 23 - 27, 1998). ICDE. IEEE Computer Society, Washington, DC, 324-331.

[66] Xu, X., Yuruk, N., Feng, Z., and Schweiger, T. A. 2007. SCAN: a structural clustering algorithm for networks. In Proceedings of the 13th ACM SIGKDD international Conference on Knowledge Discovery and Data Mining (San Jose, California, USA, August 12 - 15, 2007). KDD '07. ACM, New York, NY, 824-833. DOI= http://doi.acm.org/10.1145/1281192.1281280

[67] Yan, F., Hou, W., and Zhu, Q. 2003. Selectivity Estimation Using Orthogonal Series. In Proceedings of the Eighth international Conference on Database Systems For Advanced Applications (March 26 - 28, 2003). DASFAA. IEEE Computer Society, Washington, DC, 157.

[68] Zaki, M. J., Peters, M., Assent, I., and Seidl, T. 2007. Clicks: An effective algorithm for mining subspace clusters in categorical datasets. Data Knowl. Eng. 60, 1 (Jan. 2007), 51-70. DOI= http://dx.doi.org/10.1016/j.datak.2006.01.005

[69] Zhang, T., Ramakrishnan, R., and Livny, M. 1996. BIRCH: an efficient data clustering method for very large databases. SIGMOD Rec. 25, 2 (Jun. 1996), 103-114. DOI= http://doi.acm.org/10.1145/235968.233324

[70] Zhao, L. and Zaki, M. J. 2005. TRICLUSTER: an effective algorithm for mining coherent clusters in 3D microarray data. In Proceedings of the 2005 ACM

SIGMOD international Conference on Management of Data (Baltimore, Maryland, June 14 - 16, 2005). SIGMOD '05. ACM, New York, NY, 694-705. DOI= http://doi.acm.org/10.1145/1066157.1066236